Palm OS® Programmer’s API Reference

Palm OS® 5 SDK
# Table of Contents

## About This Document
- Palm OS SDK Documentation
- What This Volume Contains
- Additional Resources
- Conventions Used in This Guide

## Part I: User Interface

### 1 Application Launch Codes
- Launch Codes
- Launch Flags

### 2 Palm OS Events
- Event Data Structures
- Event Reference

### 3 Notifications
- Notification Data Structures
- Notification Reference

### 4 Attention Manager
- Attention Manager Data Structures
- Attention Manager Constants
- Attention Manager Functions
- Application-Defined Functions

### 5 Categories
- Category Data Structures
- Category Constants
- Category Functions

### 6 Clipboard
- Clipboard Data Structures
- Clipboard Functions
### 7 Controls
Control Data Structures ........................................ 157
Control Resources ............................................. 167
Control Functions .............................................. 167

### 8 Date and Time Selector
Date and Time Selections Data Structures ................. 185
Date and Time Selection Functions .......................... 186

### 9 Fields
Field Data Structures ........................................... 195
Field Resources ................................................. 204
Field Functions .................................................. 204

### 10 Find
Find Functions .................................................... 249

### 11 Forms
Form Data Structures ............................................ 255
Form Constants .................................................. 274
Form Resources .................................................. 274
Form Functions ................................................... 275
Application-Defined Functions ............................... 325

### 12 Graffiti Shift
GraffitiShift Functions .......................................... 331

### 13 Insertion Point
Insertion Point Functions ....................................... 335

### 14 Lists
List Data Structures ............................................ 339
List Resources ................................................... 343
List Functions .................................................... 343
Application-Defined Function ................................. 355
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Menus</td>
<td>359</td>
</tr>
<tr>
<td></td>
<td>Menu Data Structures</td>
<td>359</td>
</tr>
<tr>
<td></td>
<td>Menu Constants</td>
<td>370</td>
</tr>
<tr>
<td></td>
<td>Menu Resources</td>
<td>371</td>
</tr>
<tr>
<td></td>
<td>Menu Functions</td>
<td>371</td>
</tr>
<tr>
<td>16</td>
<td>Private Records</td>
<td>389</td>
</tr>
<tr>
<td></td>
<td>Private Record Data Structures</td>
<td>389</td>
</tr>
<tr>
<td></td>
<td>Private Record Functions</td>
<td>390</td>
</tr>
<tr>
<td>17</td>
<td>Progress Manager</td>
<td>393</td>
</tr>
<tr>
<td></td>
<td>Progress Manager Functions</td>
<td>393</td>
</tr>
<tr>
<td></td>
<td>Application-Defined Functions</td>
<td>400</td>
</tr>
<tr>
<td>18</td>
<td>Scroll Bars</td>
<td>405</td>
</tr>
<tr>
<td></td>
<td>Scroll Bar Data Structures</td>
<td>405</td>
</tr>
<tr>
<td></td>
<td>Scroll Bar Resources</td>
<td>410</td>
</tr>
<tr>
<td></td>
<td>Scroll Bar Functions</td>
<td>410</td>
</tr>
<tr>
<td>19</td>
<td>System Dialogs</td>
<td>417</td>
</tr>
<tr>
<td></td>
<td>System Dialog Functions</td>
<td>417</td>
</tr>
<tr>
<td>20</td>
<td>Tables</td>
<td>419</td>
</tr>
<tr>
<td></td>
<td>Table Data Structures</td>
<td>419</td>
</tr>
<tr>
<td></td>
<td>Table Constants</td>
<td>432</td>
</tr>
<tr>
<td></td>
<td>Table Resource</td>
<td>432</td>
</tr>
<tr>
<td></td>
<td>Table Functions</td>
<td>433</td>
</tr>
<tr>
<td></td>
<td>Application-Defined Functions</td>
<td>475</td>
</tr>
<tr>
<td>21</td>
<td>UI Color List</td>
<td>479</td>
</tr>
<tr>
<td></td>
<td>UI Color Data Types</td>
<td>479</td>
</tr>
<tr>
<td></td>
<td>UI Color Functions</td>
<td>483</td>
</tr>
<tr>
<td>22</td>
<td>UI Controls</td>
<td>489</td>
</tr>
<tr>
<td></td>
<td>UI Control Functions</td>
<td>489</td>
</tr>
</tbody>
</table>
vi  Palm OS Programmer’s API Reference
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 File Streaming</td>
<td>File Streaming Constants</td>
<td>673</td>
</tr>
<tr>
<td></td>
<td>File Streaming Functions</td>
<td>675</td>
</tr>
<tr>
<td></td>
<td>File Streaming Error Codes</td>
<td>693</td>
</tr>
<tr>
<td>32 Float Manager</td>
<td>Float Manager Data Structures</td>
<td>695</td>
</tr>
<tr>
<td></td>
<td>Float Manager Functions</td>
<td>697</td>
</tr>
<tr>
<td>33 Fonts</td>
<td>Font Data Structures</td>
<td>709</td>
</tr>
<tr>
<td></td>
<td>Font Constants</td>
<td>717</td>
</tr>
<tr>
<td></td>
<td>Font Resources</td>
<td>718</td>
</tr>
<tr>
<td></td>
<td>Font Functions</td>
<td>723</td>
</tr>
<tr>
<td>34 Graffiti Manager</td>
<td>Graffiti Manager Functions</td>
<td>737</td>
</tr>
<tr>
<td>35 Helper API</td>
<td>Helper Data Structures</td>
<td>749</td>
</tr>
<tr>
<td></td>
<td>Helper Constants</td>
<td>756</td>
</tr>
<tr>
<td>36 Key Manager</td>
<td>Key Manager Functions</td>
<td>759</td>
</tr>
<tr>
<td>37 Locale Manager</td>
<td>Locale Manager Data Types</td>
<td>763</td>
</tr>
<tr>
<td></td>
<td>Locale Manager Constants</td>
<td>765</td>
</tr>
<tr>
<td></td>
<td>Locale Manager Functions</td>
<td>768</td>
</tr>
<tr>
<td>38 Memory Manager</td>
<td>Memory Manager Functions</td>
<td>775</td>
</tr>
<tr>
<td>39 Notification Manager</td>
<td>Notification Constants</td>
<td>801</td>
</tr>
<tr>
<td></td>
<td>Notification Functions</td>
<td>802</td>
</tr>
<tr>
<td></td>
<td>Application-Defined Functions</td>
<td>810</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>48 System Event Manager</td>
<td>941</td>
<td></td>
</tr>
<tr>
<td>System Event Manager Data Structures</td>
<td>941</td>
<td></td>
</tr>
<tr>
<td>System Event Manager Functions</td>
<td>942</td>
<td></td>
</tr>
<tr>
<td>49 System Manager</td>
<td>961</td>
<td></td>
</tr>
<tr>
<td>System Manager Data Structures</td>
<td>961</td>
<td></td>
</tr>
<tr>
<td>System Functions</td>
<td>962</td>
<td></td>
</tr>
<tr>
<td>Application-Defined Functions</td>
<td>995</td>
<td></td>
</tr>
<tr>
<td>50 Text Manager</td>
<td>997</td>
<td></td>
</tr>
<tr>
<td>Text Manager Data Structures</td>
<td>997</td>
<td></td>
</tr>
<tr>
<td>Text Manager Functions</td>
<td>998</td>
<td></td>
</tr>
<tr>
<td>51 Text Services Manager</td>
<td>1041</td>
<td></td>
</tr>
<tr>
<td>Text Services Manager Data Structures</td>
<td>1041</td>
<td></td>
</tr>
<tr>
<td>Text Services Manager Functions</td>
<td>1042</td>
<td></td>
</tr>
<tr>
<td>52 Time Manager</td>
<td>1045</td>
<td></td>
</tr>
<tr>
<td>Time Manager Data Structures</td>
<td>1045</td>
<td></td>
</tr>
<tr>
<td>Time Manager Constants</td>
<td>1054</td>
<td></td>
</tr>
<tr>
<td>Time Manager Functions</td>
<td>1055</td>
<td></td>
</tr>
<tr>
<td>53 Virtual File System Manager</td>
<td>1075</td>
<td></td>
</tr>
<tr>
<td>VFS Manager Data Structures</td>
<td>1075</td>
<td></td>
</tr>
<tr>
<td>VFS Manager Constants</td>
<td>1079</td>
<td></td>
</tr>
<tr>
<td>VFS Manager Functions</td>
<td>1085</td>
<td></td>
</tr>
<tr>
<td>Application-Defined Functions</td>
<td>1145</td>
<td></td>
</tr>
<tr>
<td>54 Windows</td>
<td>1147</td>
<td></td>
</tr>
<tr>
<td>Window Data Structures</td>
<td>1147</td>
<td></td>
</tr>
<tr>
<td>Window Constants</td>
<td>1163</td>
<td></td>
</tr>
<tr>
<td>Window Functions</td>
<td>1164</td>
<td></td>
</tr>
</tbody>
</table>
Part III: Communications

56 Connection Manager

Connection Manager Data Types ........................................ 1263
Connection Manager Constants ......................................... 1263
Connection Manager Functions ......................................... 1273

57 Exchange Manager

Exchange Manager Data Structures .................................... 1297
Exchange Manager Constants ............................................ 1306
Exchange Manager Functions ............................................ 1309
Application-Defined Functions ........................................ 1352

58 Exchange Library

Exchange Library Functions ............................................. 1357

59 IR Library

IR Library Data Structures ............................................... 1373
IR Library Constants ..................................................... 1378
IR Stack Callback Events ............................................... 1380
IR Library Functions ..................................................... 1383
IAS Functions .................................................................. 1399
Application-Defined Functions ........................................ 1409

60 Modem Manager

Modem Manager Functions ................................................ 1411

61 Net Library

Net Library Data Structures ............................................... 1413
Net Library Constants ..................................................... 1420
Net Library Functions ..................................................... 1422

62 Network Utilities

Network Utility Functions .................................................. 1507
### 63 Script Plugin
- Script Plugin Data Types: 1511
- Script Plugin Constants: 1516
- Script Plugin Functions: 1518

### 64 Virtual Drivers
- Driver Data Structures: 1523
- Driver Constants: 1535
- Virtual Driver-Defined Functions: 1538
- Serial Manager Queue Functions: 1546

### 65 Serial Manager
- Serial Manager Data Structures: 1551
- Serial Manager Constants: 1557
- Serial Manager Functions: 1563
- Serial Manager Application-Defined Functions: 1590

### 66 Old Serial Manager
- Serial Manager Data Structures: 1593
- Serial Manager Functions: 1595

### 67 Serial Link Manager
- Serial Link Manager Functions: 1611

### 68 Telephony Basic Services
- Telephony Service Types: 1621
- Telephony Data Structures: 1623
- Telephony Constants: 1630
- Telephony Functions: 1637
- Feature Support Functions: 1676

### 69 Telephony Security and Configuration
- Telephony Security and Configuration Data Structures: 1681
- Telephony Security and Configuration Constants: 1684
- Telephony Security and Configuration Functions: 1684
# 70 Telephony Network
- Telephony Network Data Structures: 1695
- Telephony Network Constants: 1698
- Telephony Network Functions: 1699

# 71 Telephony Calls
- Telephony Calls Data Structures: 1715
- Telephony Calls Functions: 1720

# 72 Telephony SMS
- Telephony SMS Data Structures: 1755
- Telephony SMS Constants: 1781
- Telephony SMS Functions: 1784

# 73 Telephony Phone Book
- Telephony Phone Book Data Structures: 1815
- Telephony Phone Book Constants: 1820
- Telephony Phone Book Functions: 1821

# Part IV: Libraries

## 74 Internet Library
- Internet Library Data Structures: 1839
- Internet Library Constants: 1850
- Internet Library Functions: 1853

## 75 PalmOSGlue Library
- PalmOSGlue Functions: 1892

## 76 Bluetooth Library: General Functions
- Security Functions: 1925
- Utility Functions: 1929

## 77 Bluetooth Library: Management
- Bluetooth Management Data Structures: 1940
- Management Callback Events: 1948
- Management Event Status Codes: 1957
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Library Management Functions</td>
<td>1959</td>
</tr>
<tr>
<td></td>
<td>Management Functions</td>
<td>1962</td>
</tr>
<tr>
<td></td>
<td>Application-Defined Functions</td>
<td>1989</td>
</tr>
<tr>
<td>078</td>
<td>Bluetooth Library: Sockets and Service Discovery</td>
<td>1991</td>
</tr>
<tr>
<td></td>
<td>Socket-Related Data Structures</td>
<td>1992</td>
</tr>
<tr>
<td></td>
<td>Socket Callback Events</td>
<td>2011</td>
</tr>
<tr>
<td></td>
<td>Socket Disconnection Error Codes</td>
<td>2023</td>
</tr>
<tr>
<td></td>
<td>Socket Functions</td>
<td>2024</td>
</tr>
<tr>
<td></td>
<td>Service Discovery Protocol Functions</td>
<td>2041</td>
</tr>
<tr>
<td></td>
<td>Application-Defined Functions</td>
<td>2076</td>
</tr>
<tr>
<td>079</td>
<td>Cryptography Provider Manager</td>
<td>2079</td>
</tr>
<tr>
<td></td>
<td>The Default Provider</td>
<td>2079</td>
</tr>
<tr>
<td></td>
<td>Fundamental CPM Functions</td>
<td>2080</td>
</tr>
<tr>
<td></td>
<td>Using the Crypto-Info Structures</td>
<td>2080</td>
</tr>
<tr>
<td></td>
<td>Using the Export Functions</td>
<td>2081</td>
</tr>
<tr>
<td></td>
<td>CPM and AP Constants</td>
<td>2082</td>
</tr>
<tr>
<td></td>
<td>CPM and AP Structures and Data Types</td>
<td>2091</td>
</tr>
<tr>
<td></td>
<td>CPM Functions</td>
<td>2098</td>
</tr>
<tr>
<td></td>
<td>CPM Error Codes</td>
<td>2132</td>
</tr>
<tr>
<td>080</td>
<td>SSL Functions</td>
<td>2135</td>
</tr>
<tr>
<td></td>
<td>SSL Attribute Functions and Macros</td>
<td>2136</td>
</tr>
<tr>
<td></td>
<td>A Note on the Function Names</td>
<td>2136</td>
</tr>
<tr>
<td></td>
<td>SSL Library Functions</td>
<td>2137</td>
</tr>
<tr>
<td></td>
<td>Application-Defined Functions</td>
<td>2158</td>
</tr>
<tr>
<td>081</td>
<td>SSL Structures and Data Types</td>
<td>2163</td>
</tr>
<tr>
<td></td>
<td>SSL Data Types</td>
<td>2163</td>
</tr>
<tr>
<td></td>
<td>SSL Structures</td>
<td>2166</td>
</tr>
<tr>
<td>082</td>
<td>SSL Attributes and Macros</td>
<td>2181</td>
</tr>
<tr>
<td></td>
<td>SSL Macro Names</td>
<td>2181</td>
</tr>
<tr>
<td></td>
<td>SSL Attribute Data Types</td>
<td>2182</td>
</tr>
<tr>
<td></td>
<td>SSL Macro Pseudo-Protocol</td>
<td>2183</td>
</tr>
</tbody>
</table>
SSL Attributes ......................................................... 2187
SSL Attribute Constants ........................................... 2210

83 SSL Error Codes 2213
SSL Function Protocol Errors ................................. 2213
SSL Alerts ................................................................. 2214
SSL Handshake Errors. .............................................. 2215
SSL Cryptography Errors .......................................... 2215
SSL Illegal Message Errors ....................................... 2216
SSL Certificate Errors .............................................. 2216

84 SMS Exchange Library 2219
SMS Exchange Library Data Structures .................. 2219
SMS Exchange Library Constants ......................... 2232

85 Personal Data Interchange Library 2237
PDI Library Data Structures .................................... 2237
PDI Library Constants. ............................................. 2241
PDI Library Functions ............................................... 2253

86 Unified Data Access Manager 2279
UDA Manager Data Structures .................................. 2279
UDA Manager Constants .......................................... 2283
UDA Manager Functions ........................................... 2284
UDA Object Creation Functions ............................... 2291

Part V: Appendixes

A System Use Only Functions .................................. 2297

B Compatibility Guide 2303
2.0 New Feature Set ................................................. 2304
3.0 New Feature Set ............................................... 2308
3.1 New Feature Set ............................................... 2312
3.2 New Feature Set ............................................... 2315
International Feature Set ......................................... 2316
Japanese Feature Set ......................... 2318
Wireless Internet Feature Set ............... 2318
New Serial Manager Feature Set .......... 2320
Connection Manager Feature Set .......... 2323
3.5 New Feature Set ......................... 2324
Notification Feature Set .................... 2330
4.0 New Feature Set ......................... 2330
Expansion Manager Feature Set .......... 2336
VFS Manager Feature Set .................... 2337
Bluetooth Library Feature Set .......... 2339
High-Density Display Feature Set .......... 2342
Sound Stream Feature Set ................... 2344
5.0 New Feature Set ......................... 2346
5.1 New Feature Set ......................... 2351

C 1.0 Float Manager 2355
Float Manager Functions ..................... 2355

Index 2363
About This Document

Palm OS Programmer’s API Reference is part of the Palm OS® Software Development Kit. This introduction provides an overview of SDK documentation, discusses what materials are included in this document, and what conventions are used.

Palm OS SDK Documentation

The following documents are part of the SDK:

<table>
<thead>
<tr>
<th>Document</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Palm OS Programmer’s API Reference</td>
<td>An API reference document that contains descriptions of all Palm OS function calls and important data structures.</td>
</tr>
<tr>
<td>Palm OS Programmer’s Companion</td>
<td>A multi-volume guide to application programming for the Palm OS. This guide contains conceptual and “how-to” information that complements the Reference.</td>
</tr>
<tr>
<td>Constructor for Palm OS</td>
<td>A guide to using Constructor to create Palm OS resource files.</td>
</tr>
<tr>
<td>Palm OS Programming Development Tools Guide</td>
<td>A guide to writing and debugging Palm OS applications with the various tools available.</td>
</tr>
</tbody>
</table>

What This Volume Contains

This section provides an overview of this volume.

- Part I, “User Interface,” documents the API contained in the header files in the \Incs\Core\UI\ folder. This part contains chapters covering subjects such as application launch codes, user interface resources, events, and all window, form, and field object managers.
About This Document

Part II, “System Management,” documents the API contained in the header files in the \Incs\Core\System\ folder. This part contains chapters covering subjects such as the alarm manager, data and resource manager, feature manager, float manager, graffiti manager, key manager, memory manager, preferences manager, sound manager, string manager, and system manager.

Part III, “Communications,” documents the API related to communications, such as the exchange manager, IR library, net library, Secure Sockets Layer (SSL) library, serial manager, and serial drivers.

Part IV, “Libraries,” documents the API contained in the header files in the \Incs\Libraries\ folder. This part contains chapters covering the Internet Library and the Palm OS Glue library.

Additional Resources

• Documentation
  PalmSource publishes its latest versions of this and other documents for Palm OS developers at
  http://www.palmos.com/dev/support/docs/

• Training
  PalmSource and its partners host training classes for Palm OS developers. For topics and schedules, check
  http://www.palmos.com/dev/training

• Knowledge Base
  The Knowledge Base is a fast, web-based database of technical information. Search for frequently asked questions (FAQs), sample code, white papers, and the development documentation at
  http://www.palmos.com/dev/support/kb/
Conventions Used in This Guide

This guide uses the following typographical conventions:

<table>
<thead>
<tr>
<th>This style...</th>
<th>Is used for...</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed width font</td>
<td>Code elements such as function, structure, field, bitfield.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Emphasis (for other elements).</td>
</tr>
<tr>
<td><strong>blue and underlined</strong></td>
<td>Hot links.</td>
</tr>
</tbody>
</table>
Part I: User Interface
Application Launch Codes

This chapter provides detailed information about the predefined application launch codes. Launch codes are declared in the header file `SystemMgr.h`. The associated parameter blocks are declared in `AppLaunchCmd.h`, `AlarmMgr.h`, `ExgMgr.h`, and `Find.h`.

Table 1.1 lists all Palm OS® standard launch codes. More detailed information is provided immediately after the table:

- Launch Codes
- Launch Flags

To learn what a launch code is and how to use it, see the chapter titled “Application Startup and Stop” in the Palm OS Programmer’s Companion, vol. I.

Table 1.1 Palm OS Launch Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>scptLaunchCmdExecuteCmd</td>
<td>Execute the specified Network login script plugin command.</td>
</tr>
<tr>
<td>scptLaunchCmdListCmds</td>
<td>Provide information about the commands that your Network script plugin executes.</td>
</tr>
<tr>
<td>sysAppLaunchCmdAddRecord</td>
<td>Add a record to a database.</td>
</tr>
<tr>
<td>sysAppLaunchCmdAlarmTriggered</td>
<td>Schedule next alarm or perform quick actions such as sounding alarm tones.</td>
</tr>
<tr>
<td>sysAppLaunchCmdAttention</td>
<td>Perform the action requested by the attention manager.</td>
</tr>
</tbody>
</table>
## Application Launch Codes

Table 1.1 Palm OS Launch Codes *(continued)*

<table>
<thead>
<tr>
<th>Code</th>
<th>Request</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sysAppLaunchCmdCardLaunch</code></td>
<td>Launch the application. This launch code signifies that the application is being launched from an expansion card.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdCountryChange</code></td>
<td>Respond to country change.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdDisplayAlarm</code></td>
<td>Display specified alarm dialog or perform time-consuming alarm-related actions.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdExqAskUser</code></td>
<td>Let application override display of dialog asking user if they want to receive incoming data via the Exchange Manager.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdExqGetData</code></td>
<td>Notify application that it should send data using the Exchange Manager.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdExqPreview</code></td>
<td>Notify application that it should display a preview using the Exchange Manager.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdExqReceiveData</code></td>
<td>Notify application that it should receive incoming data using the Exchange Manager.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdFind</code></td>
<td>Find a text string.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdGoto</code></td>
<td>Go to a particular record, display it, and optionally select the specified text.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdGoToURL</code></td>
<td>Launch an application and open a URL.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdHandleSyncCallApp</code></td>
<td>Perform some application-specific operation at the behest of the application’s conduit.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdInitDatabase</code></td>
<td>Initialize database.</td>
</tr>
</tbody>
</table>
### Application Launch Codes

Table 1.1 Palm OS Launch Codes *(continued)*

<table>
<thead>
<tr>
<th>Code</th>
<th>Request</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sysAppLaunchCmdLookup</code></td>
<td>Look up data. In contrast to <code>sysAppLaunchCmdFind</code>, a level of indirection is implied. For example, look up a phone number associated with a name.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdNormalLaunch</code></td>
<td>Launch normally.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdNotify</code></td>
<td>Notify about an event.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdOpenDB</code></td>
<td>Launch application and open a database.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdPanelCalledFromApp</code></td>
<td>Tell preferences panel that it was invoked from an application, not the Preferences application.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdReturnFromPanel</code></td>
<td>Tell an application that it’s restarting after preferences panel had been called.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdSaveData</code></td>
<td>Save data. Often sent before find operations.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdSyncNotify</code></td>
<td>Notify applications that a HotSync® has been completed.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdSystemLock</code></td>
<td>Sent to the Security application to request that the system be locked down.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdSystemReset</code></td>
<td>Respond to system reset. No UI is allowed during this launch code.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdTimeChange</code></td>
<td>Respond to system time change.</td>
</tr>
<tr>
<td><code>sysAppLaunchCmdURLParams</code></td>
<td>Launch an application with parameters from the Web Clipping Application Viewer.</td>
</tr>
</tbody>
</table>
Launch Codes

This section provides supplemental information about launch
codes. For some launch codes, it lists the parameter block, which in
some cases provides additional information about the launch code.

**sysAppLaunchCmdAddRecord**

Add a record to an application’s database.

This launch code is used to add a message to the Mail or
iMessenger™ (on the Palm VII™ organizer) application’s outbox. You
pass information about the message such as address, body text, etc.
in the parameter block. For iMessenger, you can set the edit field
of the parameter block to control whether or not the iMessenger
editor is displayed. Set it to true to display the editor or false not
to display it.

For more information on sending messages via iMessenger, see
“Sending Email Messages” on page 210 in the Palm OS Programmer’s

**IMPORTANT:** Implemented for iMessenger only if Wireless
Internet Feature Set is present. Implemented for Mail only on OS
version 3.0 or later.

**sysAppLaunchCmdAddRecord Parameter Block for Mail
Application**

**Prototype**

typedef enum {
    mailPriorityHigh,
    mailPriorityNormal,
    mailPriorityLow
} MailMsgPriorityType;

typedef struct {
    Boolean secret;
    Boolean signature;
    Boolean confirmRead;
    Boolean confirmDelivery;
    MailMsgPriorityType priority;
    UInt8 padding
    Char* subject;
} MailMsgPriorityType;

Palm OS Programmer’s API Reference
Char* from;
Char* to;
Char* cc;
Char* bcc;
Char* replyTo;
Char* body;
}

Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>secret</td>
<td>True means that the message should be marked secret.</td>
</tr>
<tr>
<td>signature</td>
<td>True means that the signature from the Mail application’s preferences should be attached to the message.</td>
</tr>
<tr>
<td>confirmRead</td>
<td>True means that a confirmation should be sent when the message is read.</td>
</tr>
<tr>
<td>confirmDelivery</td>
<td>True means that a confirmation should be sent when the message is delivered.</td>
</tr>
<tr>
<td>priority</td>
<td>Message priority. Specify one of the MailMsgPriorityType enumerated types.</td>
</tr>
<tr>
<td>padding</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>subject</td>
<td>Message’s subject, a null-terminated string (optional).</td>
</tr>
<tr>
<td>from</td>
<td>Message’s sender, a null-terminated string (not used on outgoing mail).</td>
</tr>
<tr>
<td>to</td>
<td>Address of the recipient, a null-terminated string (required).</td>
</tr>
<tr>
<td>cc</td>
<td>Addresses of recipients to be copied, a null-terminated string (optional).</td>
</tr>
<tr>
<td>bcc</td>
<td>Addresses of recipients to be blind copied, a null-terminated string (optional).</td>
</tr>
<tr>
<td>replyTo</td>
<td>Reply to address, a null-terminated string (optional).</td>
</tr>
<tr>
<td>body</td>
<td>The text of the message, a null-terminated string (required).</td>
</tr>
</tbody>
</table>
sysAppLaunchCmdAddRecord Parameter Block for iMessenger Application

Prototype
typedef struct {
    UInt16 category;
    Boolean edit;
    Boolean signature;
    Char *subject;
    Char *from;
    Char *to;
    Char *replyTo;
    Char *body;
} MsgAddRecordParamsType;

Fields
category
Category in which to place the message. Specify one of the following categories:
- MsgInboxCategory
- MsgOutboxCategory
- MsgDeletedCategory
- MsgFiledCategory
- MsgDraftCategory

edit
True means that the message should be opened in the editor. False means that the message should simply be placed into the outbox and the editor not opened. You can specify true only if the category is set to MsgOutboxCategory.

signature
True means that the signature from the iMessenger application preferences should be attached to the message.

subject
Message’s subject, a null-terminated string (optional).

from
Message’s sender, a null-terminated string (not used on outgoing mail).

to
Address of the recipient, a null-terminated string (required).
replyTo  Reply to address, a null-terminated string (optional).

body  The text of the message, a null-terminated string (required).

**sysAppLaunchCmdAlarmTriggered**

Performs quick action such as scheduling next alarm or sounding alarm.

This launch code is sent as close to the actual alarm time as possible. An application may perform any quick, non-blocking action at this time. Multiple alarms may be pending at the same time for multiple applications, and one alarm display shouldn’t block the system and prevent other applications from receiving their alarms in a timely fashion. An opportunity to perform more time-consuming actions will come when `sysAppLaunchCmdDisplayAlarm` is sent.

**sysAppLaunchCmdAlarmTriggered Parameter Block**

Prototype
typedef struct SysAlarmTriggeredParamType {
  UInt32 ref;
  UInt32 alarmSeconds;
  Boolean purgeAlarm;
  UInt8 padding;
} SysAlarmTriggeredParamType;

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-&gt; ref</td>
<td>The caller-defined value specified when the alarm was set with <code>AlmSetAlarm</code></td>
</tr>
<tr>
<td>-&gt; alarmSeconds</td>
<td>The date/time specified when the alarm was set with <code>AlmSetAlarm</code>. The value is given as the number of seconds since 1/1/1904.</td>
</tr>
<tr>
<td>&lt;- purgeAlarm</td>
<td>Upon return, set to <code>true</code> if the alarm should be removed from the alarm table. Use this as an optimization to prevent the application from receiving <code>sysAppLaunchCmdDisplayAlarm</code> if you don’t wish to perform any other processing for this alarm. If you do want to receive the launch code, set this field to <code>false</code>.</td>
</tr>
</tbody>
</table>
padding  

**sysAppLaunchCmdAttention**

Perform the action requested by the attention manager. This launch code is accompanied by a value of the `AttnCommand` type; this type specifies the set of possible commands that can be sent to the application that requested the alarm.

```c
typedef UInt16 AttnCommand;
```

The following table lists the values that `AttnCommand` can assume.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttnCommand_drawDetail</td>
<td>((AttnCommand) 1)</td>
<td>Indicates that the application needs to draw the detailed contents of the attention dialog. The command arguments parameter points to a structure of type <code>AttnCommandDrawDetailArgsType</code>.</td>
</tr>
<tr>
<td>AttnCommand_drawList</td>
<td>((AttnCommand) 2)</td>
<td>Indicates that the application needs to draw the appropriate list item in the attention dialog. The command arguments parameter points to a structure of type <code>AttnCommandDrawListArgsType</code>.</td>
</tr>
</tbody>
</table>
### Application Launch Codes

**Launch Codes**

---

Table 1.2 sysAppLaunchCmdAttention Commands

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttnCommand_customEffect</td>
<td>((AttnCommand) 3)</td>
<td>Indicates that the Attention Manager is doing something to get the user's attention, and any application-specific special effect should be done. This command is only sent to attention items that set the AttnFlags_CustomEffectBit when they call AttnGetAttention, which most applications won't do.</td>
</tr>
<tr>
<td>AttnCommand_goThere</td>
<td>((AttnCommand) 4)</td>
<td>Tells the application to navigate to the item. The command arguments parameter is NULL. An application commonly calls SysAppLaunch upon receipt of this command to have itself launched.</td>
</tr>
<tr>
<td>AttnCommand_gotIt</td>
<td>((AttnCommand) 5)</td>
<td>Tells the application that the user is dismissing the item. The command arguments parameter is NULL. The application may choose to clean up memory at this point.</td>
</tr>
</tbody>
</table>
When `AttnCommand_drawDetail` is passed to the application, either via the callback function or as a parameter accompanying the `sysAppLaunchCmdAttention` launch code, the application needs to draw the detailed contents of the attention dialog. The `AttnCommandDrawDetailArgsType` structure accompanies the `AttnCommand_drawDetail` command, and provides the information needed to draw the contents of that dialog.

```c
typedef struct {
    RectangleType bounds;
    Boolean firstTime;
    AttnFlagsType flags;
} AttnCommandDrawDetailArgsType;
```
Field Descriptions

bounds

Contains the window-relative bounding box for the area to draw. The clipping region is also set to the dimensions of this box to prevent accidental drawing outside.

firstTime

Set to true if the user has not yet seen this item. The value of this field could be used, for example, to display attentions that the user hasn’t seen before in some special way.

flags

The global user preferences for this attention attempt combined with the custom flags passed in by the developer. For example, if the global preference is to mute sounds, and the developer flags are both zero, then the AttnFlags_NoSound flag is on and the AttnFlags_AlwaysSound flag is off.

AttnCommandDrawListArgsType

When AttnCommand_drawList is passed to the application, either via the callback function or as a parameter accompanying the sysAppLaunchCmdAttention launch code, the application is to draw the appropriate list item in the attention dialog. The AttnCommandDrawListArgsType structure accompanies the AttnCommand_drawList command, and provides the information needed to draw the contents of that dialog.

```c
typedef struct {
    RectangleType bounds;
    Boolean firstTime;
    AttnFlagsType flags;
} AttnCommandDrawListArgsType;
```
Field Descriptions

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bounds</td>
<td>Contains the window-relative bounding box for the area to draw. The clipping region is also set to the dimensions of this box to prevent accidental drawing outside.</td>
</tr>
<tr>
<td>firstTime</td>
<td>Set to true if the user has not yet seen this item. The value of this field could be used, for example, to trigger a custom sound the first time this attention item is presented to the user.</td>
</tr>
<tr>
<td>flags</td>
<td>The global user preferences for this attention attempt combined with the custom flags passed in by the developer. For example, if the global preference is to mute sounds, and the developer flags were both zero, then the AttnFlags_NoSound flag is on and the AttnFlags_AlwaysSound flag is off.</td>
</tr>
</tbody>
</table>

**AttnCommandGotItArgsType**

When AttnCommand_gotIt is passed to the application, either via the callback function or as a parameter accompanying the sysAppLaunchCmdAttention launch code, it is accompanied by an AttnCommandGotItArgsType structure. This structure indicates whether the AttnCommand_gotIt command was generated because the user dismissed the attention, or whether the system is simply informing your application that AttnForgetIt was called. Your application normally ignores the latter case if your application made the call to AttnForgetIt.
typedef struct {
    Boolean dismissedByUser;
} AttnCommandGotItArgsType;

Field Descriptions

dismissedByUser true indicates that the user dismissed the attention. false indicates that the AttnCommand_gotIt command was generated by a call to AttnForgetIt.

sysAppLaunchCmdCardLaunch

This launch code is sent to applications that are being run from an expansion card. The application is copied into the device’s main memory prior to being sent this launch code. If the application doesn’t respond to sysAppLaunchCmdCardLaunch, it is then sent a sysAppLaunchNormalLaunch launch code. Applications that can profit from the knowledge that they are being launched from an expansion card may want to consult the fields in the parameter block that accompanies sysAppLaunchCmdCardLaunch.

When the Launcher sends sysAppLaunchCmdCardLaunch to an application, it also sends sysAppLaunchFlagNewGlobals, and sysAppLaunchFlagUIApp flags. These two flags are not sent to start.prc, however. Applications should never interact with the user upon receiving this launch code, and should not depend on globals being available. This launch code is intended to notify the application that it is being launched from a card. Applications typically save some state information upon receiving this launch code and do the bulk of their processing when they receive sysAppLaunchNormalLaunch.

sysAppLaunchCmdCardLaunch Parameter Block

Prototype typedef struct {
    Err err;
    Uint16 volRefNum;
    const Char *path;
    Uint16 startFlags;
}
Application Launch Codes

Launch Codes

```c
} SysAppLaunchCmdCardType;
```

**Fields**

- **< span id="initial(ERR)" > err</span>**
  
  Initially set to `expErrUnsupportedOperation`, applications that recognize `sysAppLaunchCmdCardLaunch` and that don’t want to receive the subsequent `sysAppLaunchNormalLaunch` launch code should set this field to `errNone`.

- **volRefNum**
  
  The reference number of the volume from which the application is being launched.

- **path**
  
  The complete path to the application being launched.

- **startFlags**
  
  This field is made up of a combination of the following flags.

  - `sysAppLaunchStartFlagAutoStart` Indicates that the application is being run automatically upon card insertion.

  - `sysAppLaunchStartFlagNoUISwitch` Set this bit to prevent a UI switch to the application.

  - `sysAppLaunchStartFlagNoAutoDelete` Set this bit to prevent the VFS Manager from deleting the copy of the application in main memory when the associated volume is unmounted.

**sysAppLaunchCmdCountryChange**

Responds to country change.

Applications should change the display of numbers to use the proper number separators. To do this, call `LocGetNumberSeparators`, `StrLocalizeNumber`, and `StrDelocalizeNumber`.

**sysAppLaunchCmdDisplayAlarm**

Performs full, possibly blocking, handling of alarm.
This is the application’s opportunity to handle an alarm in a lengthy or blocking fashion. Alert dialogs are usually displayed when this launch code is received. This work should be done here, not when `sysAppLaunchCmdAlarmTriggered` is received. Multiple alarms may be pending at the same time for multiple applications, and one alarm display shouldn’t block the system and prevent other applications from receiving their alarms in a timely fashion.

### `sysAppLaunchCmdDisplayAlarm` Parameter Block

**Prototype**

```c
typedef struct SysDisplayAlarmParamType {
    UInt32     ref;
    UInt32     alarmSeconds;
    Boolean    soundAlarm;
    UInt8      padding;
} SysDisplayAlarmParamType;
```

**Fields**

- `-> ref` The caller-defined value specified when the alarm was set with `AlmSetAlarm`.
- `-> alarmSeconds` The date/time specified when the alarm was set with `AlmSetAlarm`. The value is given as the number of seconds since 1/1/1904.
- `-> soundAlarm` `true` if the alarm should be sounded, `false` otherwise. This value is currently not used.
- `padding` Not used.

### `sysAppLaunchCmdExgAskUser`

The Exchange Manager sends the `sysAppLaunchCmdExgAskUser` launch code to the application when data has arrived for that application. This launch code allows the application to tell the Exchange Manager not to display the exchange dialog, which it uses to have the user confirm the receipt of data. If the application does not handle this launch code, the default course of action is that the Exchange Manager displays the exchange dialog.

Applications may want to respond to this launch code under these circumstances:
• To reject all incoming data or to reject data under certain circumstances without first prompting the user. To reject incoming data, set the `result` field of the parameter block to `exgAskCancel` and then return.

• To receive incoming data without confirmation. To automatically receive incoming data, set the `result` field to `exgAskOk`.

• To provide a user confirmation dialog with extra functionality. This is described in more detail below.

Starting with Palm OS 3.5, the Exchange Manager allows applications to provide extra functionality in the exchange dialog. You can have the dialog include a category pop-up list from which the user chooses a category in which to file the incoming data. If you want to provide a category pop-up list, call the `ExgDoDialog` function in response to this launch code and pass it a database that contains the categories to be listed. See the description of that function for more information.

Applications may also bypass the call to `ExgDoDialog` altogether and provide their own dialogs.

If an application responds to this launch code, it must set the `result` field in the parameter block to the appropriate value. Possible values are:

- `exgAskDialog` Display the default exchange dialog provided by Exchange Manager.
- `exgAskOk` Accept the incoming data.
- `exgAskCancel` Reject the incoming data.

On Palm OS 3.5 or higher if you don’t use the default version of the dialog, return `exgAskOk` if the user confirmed or `exgAskCancel` if the user canceled. If you don’t set the `result` field properly, two dialogs are displayed.

**IMPORTANT:** Implemented only if [3.0 New Feature Set](#) is present.
sysAppLaunchCmdExgAskUser Parameter Block

Prototype
typedef struct {
    ExgSocketPtr          socketP;
    ExgAskResultType  result;
    UInt8                 reserved;
} ExgAskParamType;

Fields
<-> socketP  Socket pointer (see ExgSocketType)
<- result    Show dialog, auto-confirm, or auto-cancel
-> reserved  Reserved for future use

sysAppLaunchCmdExgGetData

The Exchange Manager sends the sysAppLaunchCmdExgGetData launch code when the exchange library requests data to be sent to a remote device. That is, an application on a remote device has performed an ExgGet function to request data, and the Exchange Manager has determined that the launched application should handle the request.

To respond to this launch code, applications should initiate a connection with ExgPut, use ExgSend to send the data, and call ExgDisconnect when finished.

The parameter block sent with this launch code is a pointer to the ExgSocketType structure corresponding to the Exchange Manager connection on which the data is to be sent. You pass this socket pointer to ExgPut. For more details, see the “Exchange Manager” chapter.

IMPORTANT: Implemented only if 4.0 New Feature Set is present.

sysAppLaunchCmdExgPreview

Following the launch code sysAppLaunchCmdExgAskUser, the Exchange Manager sends the sysAppLaunchCmdExgPreview launch code to have the application display the preview in the exchange dialog.
Application Launch Codes

Launch Codes

sysAppLaunchCmdExgPreview Parameter Block

Prototype
typedef struct {
    Uint16             version;
    ExgSocketType      *socketP;
    Uint16             op;
    Char               *string;
    Uint32             size;
    RectangleType      bounds;
    Uint16             types;
    Err                error;
} ExgPreviewInfoType;

Fields

- `version`  
  Set this field to 0 to specify version 0 of this structure.

- `socketP`  
  A pointer to the socket structure (see ExgSocketType). The libraryRef field must point to the exchange library from which preview data should be received.

- `op`  
  A constant that identifies the operation. This can be one of the following:

  - `exgPreviewDialog`  
    Display a form or modal dialog containing the preview. This constant is only used in situations where one application launches another to display data.

  - `exgPreviewDraw`  
    Draw the preview as a graphic in the bounds rectangle.

  - `exgPreviewLongString`  
    Return the preview as a long string in the string field.

  - `exgPreviewQuery`  
    Return the list of preview modes the application supports in the types field.

  - `exgPreviewLongString`  
    Return the preview as a short string in the string field.
### Application Launch Codes

#### Launch Codes

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;string&gt;</code></td>
<td>A buffer into which the preview string is placed if one of the string preview operations is specified.</td>
</tr>
<tr>
<td><code>&gt;size</code></td>
<td>The allocated size of the <code>string</code> field.</td>
</tr>
<tr>
<td><code>&gt;bounds</code></td>
<td>The bounds of the rectangle in which to draw the graphic if the preview operation is <code>exgPreviewDraw</code>.</td>
</tr>
<tr>
<td><code>&lt;types&gt;</code></td>
<td>Upon return from <code>exgPreviewQuery</code>, a bit field identifying the types of previews the library supports (<code>exgPreviewDraw</code>, <code>exgPreviewLongString</code>, or <code>exgPreviewShortString</code>).</td>
</tr>
<tr>
<td><code>&lt;error&gt;</code></td>
<td>The error code returned from the library. If this is <code>errNone</code>, the preview operation was successful.</td>
</tr>
</tbody>
</table>

Applications that respond to this launch code should check the parameter block’s `op` field and respond as described above.

Applications can define and use their own constants for the preview operation. Operations specific to an application are numbered starting at `exgPreviewFirstUser` and should be no greater than `exgPreviewLastUser`.

Applications respond to this launch in much the same way they respond to `sysAppLaunchCmdExgReceiveData`. Use `ExgAccept` to accept the preview connection, `ExgReceive` to receive the data, and then `ExgDisconnect` to end the connection. The only difference is what is done with the data when it is received. With this launch code, the application should return the data in the `string` field or draw it in the `bounds` rectangle. With the `sysAppLaunchCmdExgReceiveData` launch code, the application stores the received data.

**IMPORTANT:** Implemented only if [4.0 New Feature Set](#) is present.
**sysAppLaunchCmdExgReceiveData**

The Exchange Manager sends the `sysAppLaunchCmdExgReceiveData` launch code following the `sysAppLaunchCmdExgAskUser` and `sysAppLaunchCmdExgPreview` launch codes to notify the application that it should receive the data (assuming that the application and/or the user has indicated the data should be received).

The application should use Exchange Manager functions to receive the data and store it or do whatever it needs to with the data. Specifically, most applications should respond to this launch code by calling `ExgAccept` to accept the connection and then `ExgReceive` to receive the data.

Note that the application may not be the active application, and thus may not have globals available when it is launched with this launch code. You can check if you have globals by using this code in the `PilotMain` routine:

```c
Boolean appIsActive = launchFlags & sysAppLaunchFlagSubCall;
```

The `appIsActive` value is true if your application is active and globals are available; otherwise, you won’t be able to access any of your global variables during the receive operation.

The parameter block sent with this launch code is a pointer to the `ExgSocketType` structure corresponding to the Exchange Manager connection on which the data is arriving. Pass this pointer to the `ExgAccept` function to begin receiving the data. For more details, refer to the “Exchange Manager” chapter.

**IMPORTANT**: Implemented only if 3.0 New Feature Set is present.

**sysAppLaunchCmdFind**

This launch code is used to implement the global find. When the user enters a text string in the Find dialog, the system sends this launch code with the `FindObjectType` parameter block to each
application. The application should search for the string that the user entered and return any records matching the find request.

The system displays the results of the query in the Find results dialog. The system continues the search with each application until it has a full screen of matching records or until all of the applications on the device have had a chance to respond. If the screen is full, a Find More button appears at the bottom of the dialog. If the user clicks the Find More button, the search resumes. Applications can test whether the current find is a continuation of a previous one by checking the continuation field in the parameter block.

Most applications that use text records should support this launch code. When they receive it, they should search all records for matches to the find string and return all matches. Functions that you can use to respond to this launch code are described in Chapter 10, “Find.”

An application can also integrate the find operation in its own user interface and send the launch code to a particular application.

Applications that support this launch code should support sysAppLaunchCmdSaveData and sysAppLaunchCmdGoto as well.

**sysAppLaunchCmdFind Parameter Block**

```c
Prototype typedef struct {
    UInt16    dbAccesMode;
    UInt16    recordNum;
    Boolean   more;
    Char      strAsTyped [maxFindStrLen+1];
    Char      strToFind [maxFindStrLen+1];
    UInt8     reserved1;
    UInt16    numMatches;
    UInt16    lineNumber;
    Boolean   continuation;
    Boolean   searchedCaller;
    LocalID   callerAppDbID;
    UInt16    callerAppCardNo;
    LocalID   appDbID;
    UInt16    appCardNo;
    Boolean   newSearch;
    UInt8     reserved2;
    DmSearchStateType   searchState;
}
```
FindMatchType match [maxFinds];
} FindParamsType;

**Fields**

- **dbAccesMode**  
  Mode in which to open the application’s database. Pass this directly to `DmOpenDatabase` as the mode parameter. Its value is either `dmModeReadOnly` or `dmModeReadOnly | dmModeShowSecret`. (See “Open Mode Constants” for more information.)

- **recordNum**  
  Index of last record that contained a match. Start the search from this location. Do not set this value directly. Instead, call `FindSaveMatch` when you have a matching record.

- **more**  
  If true, the Find results dialog displays the Find More button indicating that there may be more results to display.

  Typically `FindSaveMatch` handles setting the more field. Applications with large databases to search may want to periodically check for and stop the search if an event is pending. If so, they should set this field to true before stopping.

- **strAsTyped**  
  Search string as the user entered it.

- **strToFind**  
  Normalized version of the search string. The method by which a search string is normalized varies depending on the version of Palm OS and the character encoding supported by the device. You pass strToFind directly to the search function (either `FindStrInStr`, `TxtFindString`, or `TxtGlueFindString`).

- **reserved1**  
  Reserved for future use.

- **numMatches**  
  The current number of matches. Do not set this field directly. Instead, call `FindSaveMatch`, which increments it for you.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lineNumber</td>
<td>Line number of the next line that displays the results. Do not set this field directly. It is incremented by a call to <code>FindDrawHeader</code>.</td>
</tr>
<tr>
<td>continuation</td>
<td>If <code>true</code>, the launch code is being sent as part of a continuation of a previous Find. If <code>false</code>, this is a new Find. Do not set this field directly; the system sets it when the Find results dialog is full.</td>
</tr>
<tr>
<td>searchedCaller</td>
<td>If <code>true</code>, the application that was active at the time the user tapped the Find button has responded to this launch code. This application is always searched before any others.</td>
</tr>
<tr>
<td>callerAppDbID</td>
<td>Database ID of the application that was active when the user tapped the Find button. Do not change the value of this field; the system sets it and uses it when searching for application databases.</td>
</tr>
<tr>
<td>callerAppCardNo</td>
<td>Card number of the application that was active when the user tapped the Find button. Do not change the value of this field; the system sets it and uses it when searching for application databases.</td>
</tr>
<tr>
<td>appDbID</td>
<td>The ID of your application’s resource database. Do not set this field directly; the system sets it and uses it when searching for application databases.</td>
</tr>
<tr>
<td>appCardNo</td>
<td>The card number of your application’s resource database. Do not set this field directly; the system sets it and uses it when searching for application databases.</td>
</tr>
<tr>
<td>newSearch</td>
<td>System use only.</td>
</tr>
<tr>
<td>reserved2</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>searchState</td>
<td>System use only.</td>
</tr>
<tr>
<td>match</td>
<td>System use only.</td>
</tr>
</tbody>
</table>
sysAppLaunchCmdGoto

Sent in conjunction with sysAppLaunchCmdFind or sysAppLaunchCmdExgReceiveData to allow users to actually inspect the record that the global find returned or that was received by the Exchange Manager.

Applications should do most of the normal launch actions, then display the requested item. The application should continue running unless explicitly closed.

An application launched with this code does have access to global variables, static local variables, and code segments other than segment 0 (in multi-segment applications).

Applications that receive this launch code should test the sysAppLaunchFlagNewGlobals launch flag to see if they need to initialize global variables. sysAppLaunchFlagNewGlobals indicates that the system has just allocated your global variables.

For example:

```c
    case sysAppLaunchCmdGoTo:
        if (launchFlags & sysAppLaunchFlagNewGlobals)
            StartApplication();
```

Note that you shouldn’t automatically initialize the global variables in response to this launch code. Test the launch flag first. Your application receives this launch code when the user selects a record in the global find results. If your application was the current application before the user selected the Find command, the launch flag is clear to indicate that your globals should not be re-initialized.

sysAppLaunchCmdGoto Parameter Block

Prototype

typedef struct {
    Int16      searchStrLen;
    UInt16     dbCardNo;
    LocalID    dbID;
    UInt16     recordNum;
    UInt16     matchPos;
    UInt16     matchFieldNum;
    UInt32     matchCustom;
} GoToParamsType;
Fields

- **searchStrLen**: Length of normalized search string. This is **not** the length of the matching string. See below for a full explanation.
- **dbCardNo**: Card number of the database to open.
- **dbID**: Local ID of the database to open.
- **recordNum**: Index of record containing a match.
- **matchPos**: Position of the match within the field.
- **matchFieldNum**: Field number string was found in.
- **matchCustom**: Application-specific information.

Often, applications highlight the search string when displaying the resulting record. Localizable applications commonly store the length of the string to select in the `matchCustom` field for this purpose. Some multi-byte character encodings represent certain characters both as a single-byte character and a multi-byte character. When the search is performed, the single-byte character is accurately matched against its multi-byte equivalent. For this reason, the length of the string searched for does not always equal the length of the matching string. Applications that support being localized to multi-byte character sets often set the `matchCustom` field to the length of the matching string in the call to `FindSaveMatch` so that they know the length of the string to select.

**sysAppLaunchCmdGoToURL**

Applications can respond to this launch code to retrieve and display the specified URL.

The parameter block for this launch command is simply a pointer to a string containing the URL.

This launch code may be sent in the following instances:

- If the [Wireless Internet Feature Set](#) is, applications can send this launch code directly to the Web Clipping Application Viewer application.
- If [4.0 New Feature Set](#) is present, the `ExqRequest` function launches an application with this launch code if it cannot find an exchange library that is registered for the URL it has
received. To receive the launch code, the application must first use `ExgRegisterDatatype` to register for a URL scheme.

**IMPORTANT:** Implemented only if [Wireless Internet Feature Set](#) is present.

### sysAppLaunchCmdHandleSyncCallApp

This launch command is sent by the Desktop Link server when `SyncCallRemoteModule` is called from a conduit to request that the handheld application do some processing on the conduit’s behalf.

Along with this launch code you receive a `sysAppLaunchCmdHandleSyncCallApp` parameter block which contains all of the information passed to `SyncCallRemoteModule` on the desktop plus the fields needed to pass the result back to the desktop. Pass the results back to the conduit by calling `DlkControl`. See the comments section for `DlkControl` in the [Palm OS Programmer’s API Reference](#) for an example of how to handle this launch code.
sysAppLaunchCmdHandleSyncCallApp Parameter Block

Prototype

typedef struct
SysAppLaunchCmdHandleSyncCallAppType {
    UInt16 pbSize;
    UInt16 action;
    void *paramP;
    UInt32 dwParamSize;
    void *dlRefP;
    Boolean handled;
    UInt8 reserved1;
    Err replyErr;
    UInt32 dwReserved1;
    UInt32 dwReserved2;
} SysAppLaunchCmdHandleSyncCallAppType;

Fields

- **pbSize**: Size, in bytes, of this parameter block. Set to sizeof(SysAppLaunchCmdHandleSyncCallAppType).
- **action**: Call action ID (application-specific).
- **paramP**: Pointer to parameter block (call action ID specific).
- **dwParamSize**: Parameter block size, in bytes.
- **dlRefP**: DesktopLink reference pointer. Supply this value in the DlkCallAppReplyParamType structure when calling DlkControl with the dlkCtlSendCallAppReply control code.
- **handled**: Initialized to false by DLServer; if handled, your application must set it to true (and your handler the handler must call DlkControl with the dlkCtlSendCallAppReply control code). If your handler is not going to send a reply back to the conduit, leave this field set to false, in which case the DesktopLink Server will send the default "unknown request" reply.
- **reserved1**: Reserved. Set to NULL.
replyErr Error code returned from the call to DlkControl with the dlkCtlSendCallAppReply control code.

dwReserved1 Reserved. Set to NULL.

dwReserved2 Reserved. Set to NULL.

**sysAppLaunchCmdInitDatabase**

This launch code is sent by the Desktop Link server in response to a request to create a database. It is sent to the application whose creator ID matches that of the requested database.

The most frequent occurrence of this is when a 'data' database is being installed or restored from the desktop. In this case, HotSync creates a new database on the device and passes it to the application via a sysAppLaunchCmdInitDatabase command, so that the application can perform any required initialization. HotSync will then transfer the records from the desktop database to the device database.

When a Palm OS application crashes while a database is installed using HotSync, the reason may be that the application is not handling the sysAppLaunchCmdInitDatabase command properly. Be especially careful not to access global variables.

The system will create a database and pass it to the application for initialization. The application must perform any initialization required, then pass the database back to the system, unclosed.

**sysAppLaunchCmdInitDatabase Parameter Block**

Prototype

```c
typedef struct {
    DmOpenRef     dbP;
   _UINT32        creator;
   _UINT32        type;
   _UINT16        version;
} SysAppLaunchCmdInitDatabaseType;
```

**Fields**

- **dbP** Database reference.
- **creator** Database creator.
- **type** Database type.
version             Database version.

**sysAppLaunchCmdLookup**

The system or an application sends this launch command to retrieve information from another application. In contrast to Find, there is a level of indirection; for example, this launch code could be used to retrieve the phone number based on input of a name.

This functionality is currently supported by the standard Palm OS Address Book.

Applications that decide to handle this launch code must search their database for the string the user entered and perform the match operation specified in the launch code’s parameter block.

If an application wants to allow its users to perform lookup in other applications, it has to send it properly, including all information necessary to perform the match. An example for this is in Address.c and AppLaunchCmd.h, which are included in your SDK.

**sysAppLaunchCmdLookup Parameter Block**

The parameter block is defined by the application that supports this launch code. See AppLaunchCmd.h for an example.

**IMPORTANT:** Implemented only if 2.0 New Feature Set is present.

**sysAppLaunchCmdNotify**

The system or an application sends this launch code to notify applications that an event has occurred. The parameter block specifies the type of event that occurred, as well as other pertinent information. To learn which notifications are broadcast by the system, see the chapter titled “Notifications” in this book.

**IMPORTANT:** Implemented only if Notification Feature Set is present.
Application Launch Codes

Launch Codes

sysAppLaunchCmdNotify Parameter Block

The `SysNotifyParamType` structure declared in `NotifyMgr.h` defines the format of this launch code’s parameter block. See its description in the “Notifications” chapter.

sysAppLaunchCmdOpenDB

You can send this launch code to the Web Clipping Application Viewer application to launch the application and cause it to open and display a Palm™ query application stored on the device. This is the same mechanism that the Launcher uses to launch query applications.

**IMPORTANT:** Implemented only if [Wireless Internet Feature Set](#) is present.

sysAppLaunchCmdOpenDB Parameter Block

Prototype

```c
typedef struct {
    UInt16     cardNo;
    LocalID    dbID;
} SysAppLaunchCmdOpenDBType;
```

**Fields**

- `cardNo` Card number of database to open.
- `dbID` Database id of database to open.

sysAppLaunchCmdPanelCalledFromApp

`sysAppLaunchCmdPanelCalledFromApp` and `sysAppLaunchCmdReturnFromPanel` allow an application to let users change preferences without switching to the Preferences application. For example, for the calculator, you may launch the Formats preferences panel, set up a number format preference, then directly return to the calculator that then uses the new format.

`sysAppLaunchCmdPanelCalledFromApp` lets a preferences panel know whether it was switched to from the Preferences application or whether an application invoked it to make a change. The panel may be a preference panel owned by the application or a system preferences panel.
Examples of these system panels that may handle this launch code are:

- Network panel (called from network applications)
- Modem panel (called if modem selection is necessary)

All preferences panels must handle this launch code. If a panel is launched with this command, it should:

- Display a Done button.
- **Not** display the panel-switching pop-up trigger used for navigation within the preferences application.

**IMPORTANT:** Implemented only if 2.0 New Feature Set is present.

**sysAppLaunchCmdReturnFromPanel**

This launch code is used in conjunction with `sysAppLaunchCmdPanelCalledFromApp`. It informs an application that the user is done with a called preferences panel. The system passes this launch code to the application when a previously-called preferences panel exists.

**IMPORTANT:** Implemented only if 2.0 New Feature Set is present.

**sysAppLaunchCmdSaveData**

Instructs the application to save all current data. For example, before the system performs a global find, an application should save all data.

Any application that supports the Find command and that can have buffered data should support this launch code. The system sends this launch code to the currently active application before it begins the search. The application receiving this launch code should respond by saving all buffered data so that the search is able to find matches in the text just entered.
Application Launch Codes

Launch Codes

sysAppLaunchCmdSaveData Parameter Block

Prototype
typedef struct {
    Boolean    uiComing;
    UInt8      reserved1;
} SysAppLaunchCmdSaveDataType;

Fields
uiComing  true if the system dialog is displayed before launch code arrives.
reserved1  Reserved for future use.

sysAppLaunchCmdSyncNotify
This launch code is sent to applications to inform them that a HotSync operation has occurred.

This launch code is sent only to applications whose databases were changed during the HotSync operation. (Installing the application database itself is considered a change.) The record database(s) must have the same creator ID as the application in order for the system to know which application to send the launch code to.

This launch code provides a good opportunity to update, initialize, or validate the application’s new data, such as resorting records, setting alarms, and so on.

Because applications only receive sysAppLaunchCmdSyncNotify when their databases are updated, this launch code is not a good place to perform any operation that must occur after every HotSync operation. Instead, you may register to receive the sysNotifySyncFinishEvent on systems that have the Notification Feature Set. This notification is sent at the end of a HotSync operation, and it is sent to all applications registered to receive it, whether the application’s data changed or not. Note that there is also a sysNotifySyncStartEvent notification.

sysAppLaunchCmdSystemLock
Launch code sent to the system-internal security application to lock the device.
As a rule, applications don’t need to do respond to this launch code. If an application replaces the system-internal security application, it must handle this launch code.

**IMPORTANT:** Implemented only if 2.0 New Feature Set is present.

### sysAppLaunchCmdSystemReset

Launch code to respond to system soft or hard reset.

Applications can respond to this launch code by performing initialization, indexing, or other setup that they need to do when the system is reset. For more information about resetting the device, see “System Boot and Reset” in the *Palm OS Programmer’s Companion*, vol. I.

**sysAppLaunchCmdSystemReset Parameter Block**

**Prototype**

```c
typedef struct {
    Boolean    hardReset;
    Boolean    createDefaultDB;
} SysAppLaunchCmdSystemResetType;
```

**Fields**

- `hardReset`  
  true if system was hardReset. false if system was softReset.

- `createDefaultDB`  
  If true, application has to create default database.

### sysAppLaunchCmdTimeChange

Launch code to respond to a time change initiated by the user.

Applications that are dependent on the current time or date need to respond to this launch code. For example, an application that sets alarms may want to cancel an alarm or set a different one if the system time changes.

On systems that have the Notification Feature Set, applications should register to receive the `sysNotifyTimeChangeEvent`
notification instead of responding to this launch code. The syAppLaunchCmdTimeChange launch code is sent to all applications. The sysNotifyTimeChangeEvent notification is sent only to applications that have specifically registered to receive it, making it more efficient than syAppLaunchCmdTimeChange.

**sysAppLaunchCmdURLParams**

This launch code is sent from the Web Clipping Application Viewer application to launch another application.

The parameter block consists of a pointer to a special URL string, which the application must know how to parse. The string is the URL used to launch the application and may contain encoded parameters.

An application launched with this code may or may not have access to global variables, static local variables, and code segments other than segment 0 (in multi-segment applications). It depends on the URL that caused the Web Clipping Application Viewer to send this launch code. If this launch code results from a palm URL, then globals are available. If the launch code results from a palmcall URL, then globals are not available.

The best way to test if you have global variable access is to test the sysAppLaunchFlagNewGlobals launch flag sent with this launch code. If this is flag is set, then you have global variable access.

**IMPORTANT:** Implemented only if Wireless Internet Feature Set is present.

---

**Launch Flags**

When an application is launched with any launch command, it also is passed a set of launch flags.

An application may decide to ignore the flags even if it handles the launch code itself. For applications that decide to use the launch flags, the following table provides additional information:
Applications should never set launch flags when sending a launch code to another application. They should only be set by the system. In particular, note that you should never pass `sysAppLaunchFlagNewGlobals` as a launch flag for `SysAppLaunch`. If you do and you make repeated calls to `SysAppLaunch`, the system eventually runs out of owner IDs, and the new application fails to launch.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sysAppLaunchFlagNewGlobals</code></td>
<td>Set when the system has created and initialized a new globals world for the application. Implies new owner ID for memory chunks.</td>
</tr>
<tr>
<td><code>sysAppLaunchFlagUIApp</code></td>
<td>Set when a UI application is being launched.</td>
</tr>
<tr>
<td><code>sysAppLaunchFlagSubCall</code></td>
<td>Set when the application is calling its entry point as a subroutine call. This tells the launch code that it’s OK to keep the A5 (globals) pointer valid through the call. If this flag is set, it indicates that the application is already running as the current application.</td>
</tr>
</tbody>
</table>
Palm OS® events are structures (defined in the header files Event.h, SysEvent.h, and INetMgr.h) that the system passes to the application when the user interacts with the graphical user interface. Chapter 3, “Event Loop,” on page 53, in the Palm OS Programmer’s Companion, vol. 1 discusses in detail how this works. This chapter provides reference-style information about each event. First it shows the types used by Palm OS events. Then it discusses the following events in alphabetical order:

<table>
<thead>
<tr>
<th>Event</th>
<th>UI Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>appStopEvent</td>
<td>N.A.</td>
</tr>
<tr>
<td>ctlEnterEvent, ctlExitEvent, ctlRepeatEvent, ctlSelectEvent</td>
<td>Control</td>
</tr>
<tr>
<td>daySelectEvent</td>
<td>N.A.</td>
</tr>
<tr>
<td>fldChangedEvent, fldEnterEvent, fldHeightChangedEvent</td>
<td>Field</td>
</tr>
<tr>
<td>frmCloseEvent, frmGotoEvent, frmLoadEvent, frmOpenEvent, frmSaveEvent, frmUpdateEvent, frmTitleEnterEvent, frmTitleSelectEvent</td>
<td>Form</td>
</tr>
<tr>
<td>frmGadgetEnterEvent, frmGadgetMiscEvent</td>
<td>Extended gadget</td>
</tr>
<tr>
<td>inetSockReadyEvent, inetSockStatusChangeEvent</td>
<td>N.A. (INetLib)</td>
</tr>
<tr>
<td>keyDownEvent</td>
<td>N.A.</td>
</tr>
<tr>
<td>lstEnterEvent, lstExitEvent, lstSelectEvent</td>
<td>List</td>
</tr>
<tr>
<td>menuEvent, menuOpenEvent, menuCloseEvent, menuCmdBarOpenEvent</td>
<td>Menu</td>
</tr>
<tr>
<td>nilEvent</td>
<td>N.A.</td>
</tr>
<tr>
<td>penDownEvent, penMoveEvent, penUpEvent</td>
<td>N.A. (pen)</td>
</tr>
</tbody>
</table>
eventsEnum

The eventsEnum enum specifies the possible event types.

```c
enum events {
    nilEvent = 0,
    penDownEvent,
    penUpEvent,
    penMoveEvent,
    keyDownEvent,
    winEnterEvent,
    winExitEvent,
    ctlEnterEvent,
    ctlExitEvent,
    ctlSelectEvent,
    ctlRepeatEvent,
    lstEnterEvent,
    lstSelectEvent,
    lstExitEvent,
    popSelectEvent,
    fldEnterEvent,
    fldHeightChangedEvent,
    fldChangedEvent,
    tblEnterEvent,
    tblSelectEvent,
    daySelectEvent,
    menuEvent,
    appStopEvent = 22,
    frmLoadEvent,
};
```

### Event Data Structures

<table>
<thead>
<tr>
<th>Event</th>
<th>UI Object</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>popSelectEvent</code></td>
<td>Popup (Control)</td>
</tr>
<tr>
<td><code>sclEnterEvent</code>, <code>sclRepeatEvent</code>, <code>sclExitEvent</code></td>
<td>Scroll bar</td>
</tr>
<tr>
<td><code>tblEnterEvent</code>, <code>tblExitEvent</code>, <code>tblSelectEvent</code></td>
<td>Table</td>
</tr>
<tr>
<td><code>winEnterEvent</code>, <code>winExitEvent</code></td>
<td>Window</td>
</tr>
</tbody>
</table>
Each of these event types is discussed in alphabetical order below.

**EventType**

The EventType structure contains all the data associated with a system event. All event types have some common data. Most events also have data specific to those events. The specific data uses a union that is part of the EventType data structure. The union can have up to 8 words of specific data.

The common data is documented below the structure. The Event Reference section gives details on the important data associated with each type of event.
typedef struct {
    eventsEnum    eType;
    Boolean       penDown;
    UInt8         tapCount;
    Int16         screenX;
    Int16         screenY;
    union{
        ...
    } data;
} EventType;

Common Field Descriptions

eType One of the eventsEnum constants. Specifies the type of the event.

penDown true if the pen was down at the time of the event, otherwise false.

tapCount The number of taps received at this location. This value is used mainly by fields. When the user taps in a text field, two taps selects a word, and three taps selects the entire line.

screenX Window-relative position of the pen in pixels (number of pixels from the left bound of the window).

screenY Window-relative position of the pen in pixels (number of pixels from the top left of the window).

data The specific data for an event, if any. The data is a union, and its exact contents depend on the eType field. The Event Reference section in this chapter shows what the data field contains for each event.
**NOTE:** Remember that the `data` field is part of the access path to an identifier in the `EventType` structure. As an example, the code to access the `controlID` field of a `ctlEnterEvent` would be:

```c
EventType *event;
//...
if (event->data.ctlEnter.controlID == MyAppLockButton)
```

**Compatibility**

The `tapCount` field is only defined if the 3.5 New Feature Set is present. Because of the `tapCount` field, it’s particularly important that you clear the event structure before you use it to add a new event to the queue in Palm OS 3.5 and higher. Otherwise, the `tapCount` value may be incorrect for the new event.

**EventPtr**

The `EventPtr` defines a pointer to an `EventType`.

```c
typedef EventType *EventPtr;
```

**Event Reference**

**appStopEvent**

When the system wants to launch a different application than the one currently running, the event manager sends this event to request the current application to terminate. In response, an application has to exit its event loop, close any open files and forms, and exit.

If an application doesn’t respond to this event by exiting, the system can’t start the other application.
Palm OS Events
Event Reference

**ctlEnterEvent**

The control routine `CtlHandleEvent` sends this event when it receives a `penDownEvent` within the bounds of a control.

For this event, the data field contains the following structure:

```c
struct ctlEnter {
    UInt16 controlID;
    struct ControlType *pControl;
} ctlEnter;
```

**Field Descriptions**

- **controlID**  Developer-defined ID of the control.
- **pControl**  Pointer to a control structure (`ControlType`).

**ctlExitEvent**

The control routine `CtlHandleEvent` sends this event. When `CtlHandleEvent` receives a `ctlEnterEvent`, it tracks the pen until the pen is lifted from the display. If the pen is lifted within the bounds of a control, a `ctlSelectEvent` is added to the event queue; if not, a `ctlExitEvent` is added to the event queue. The `penDown`, `screenX`, and `screenY` fields of the `EventType` structure are set appropriately for the `ctlExitEvent`. As well, the data field contains the following structure:

```c
struct ctlExit {
    UInt16 controlID;
    struct ControlType *pControl;
} ctlExit;
```

**Field Descriptions**

- **controlID**  Developer-defined ID of the control.
- **pControl**  Pointer to a control structure (`ControlType`).

**ctlRepeatEvent**

The control routine `CtlHandleEvent` sends this event. When `CtlHandleEvent` receives a `ctlEnterEvent` in a repeating
button (tREP) or a feedback slider control (tslf), it sends a ctlRepeatEvent. When CtlHandleEvent receives a ctlRepeatEvent in a repeating button, it sends another ctlRepeatEvent if the pen remains down within the bounds of the control for 1/2 second beyond the last ctlRepeatEvent.

When CtlHandleEvent receives a ctlRepeatEvent in a feedback slider control, it sends a ctlRepeatEvent each time the slider’s thumb moves by at least one pixel. Feedback sliders do not send ctlRepeatEvents at regular intervals like repeating buttons do.

If you return true in response to a ctlRepeatEvent, it stops the ctlRepeatEvent loop. No further ctlRepeatEvents are sent.

For this event, the data field contains the following structure:

```c
struct ctlRepeat {
    UInt16 controlID;
    struct ControlType *pControl;
    UInt32 time;
    UInt16 value;
} ctlRepeat;
```

**Field Descriptions**

- **controlID**  
  Developer-defined ID of the control.
- **pControl**  
  Pointer to a control structure (ControlType).
- **time**  
  System-ticks count when the event is added to the queue.
- **value**  
  Current value if the control is a feedback slider.

**Compatibility**

The value field is only present if 3.5 New Feature Set is present.

**ctlSelectEvent**

The control routine CtlHandleEvent sends this event. When CtlHandleEvent receives a ctlEnterEvent, it tracks the pen until the pen is lifted. If the pen is lifted within the bounds of the same control it went down in, a ctlSelectEvent is added to the event queue; if not, a ctlExitEvent is added to the event queue.
It usually doesn’t matter whether you return `true` or `false` from your event handler since the operating system doesn’t handle this event. The default event handler for popup triggers does handle this event, however, so you must return `false` in this instance to ensure that the list is actually displayed.

For this event, the `data` field contains the following structure:

```c
struct ctlSelect {
    UInt16  controlID;
    struct ControlType *pControl;
    Boolean on;
    UInt8   reserved1;
    UInt16  value;
} ctlSelect;
```

**Field Descriptions**

- `controlID` Developer-defined ID of the control.
- `pControl` Pointer to a control structure (`ControlType`).
- `on` `true` when the control is depressed; otherwise, `false`.
- `reserved1` Unused.
- `value` Current value if the control is a slider.

**Compatibility** The `value` field is only present if 3.5 New Feature Set is present.

**daySelectEvent**

The system-internal `DayHandleEvent` routine, which handles events in the day selector object, handles this event. When the day selector object displays a calendar month, the user can select a day by tapping on it.

This event is sent when the pen touches and is lifted from a day number.

For this event, the `data` field contains the following structure:

```c
struct daySelect {
    struct DaySelectorType *pSelector;
```
Int16 selection;
Boolean useThisDate;
UInt8 reserved1;
} daySelect;

Field Descriptions

pSelector Pointer to a day selector structure (DaySelectorType).

selection Not used.

useThisDate Set to true to automatically use the selected date.

reserved1 Unused.

fldChangedEvent

The field routine FldHandleEvent sends this event when the text of a field has or might have been scrolled. This event actually can be triggered from any call to the field code that causes scrolling to happen; most often, this happens during FldHandleEvent.

When FldHandleEvent receives a fldEnterEvent, it positions the insertion point and tracks the pen until it’s lifted. Text is selected (highlighted) appropriately as the pen is dragged.

For this event, the data field contains the following structure:

    struct fldChanged {
        UInt16 fieldID;
        struct FieldType *pField;
    } fldChanged;

Field Descriptions

fieldID Developer-defined ID of the field.

pField Pointer to a field structure (FieldType).

fldEnterEvent

The field routine FldHandleEvent sends this event when the field receives a penDownEvent within the bounds of a field. For this event, the data field contains the following structure:


struct fldEnter {
    UInt16 fieldID;
    struct FieldType *pField;
} fldEnter;

Field Descriptions

fieldID  Developer-defined ID of the field.
pField   Pointer to a field structure (FieldType).

fldHeightChangedEvent

Several field routines, including FldHandleEvent, send this event when the number of lines in the field changes. These functions send a fldHeightChangedEvent to notify your application that the height of a field needs to change.

If the field is contained in a table, the table’s code handles the fldHeightChangedEvent. If the field is directly on a form, your application code should handle the fldHeightChangedEvent itself. The form code does not handle the event for you.

For this event, the data field contains the following structure:

struct fldHeightChanged {
    UInt16   fieldID;
    struct FieldType *pField;
    Int16    newHeight;
    UInt16   currentPos;
} fldHeightChanged;

Field Descriptions

fieldID  Developer-defined ID of the field.
pField   Pointer to a field structure (FieldType).
newHeight New visible height of the field, in number of lines.
currentPos Current position of the insertion point.
frmCloseEvent

The form routines FrmGotoForm and FrmCloseAllForms send this event. FrmGotoForm sends a frmCloseEvent to the currently active form; FrmCloseAllForms sends a frmCloseEvent to all forms an application has loaded into memory. If an application doesn’t intercept this event, the routine FrmHandleEvent erases the specified form and releases any memory allocated for it.

For this event, the data field contains the following structure:

```c
struct frmClose {
    UInt16 formID;
} frmClose;
```

**Field Descriptions**

- **formID**  
  Developer-defined ID of the form.

frmGadgetEnterEvent

The function FrmHandleEvent sends this event when there is a penDownEvent within the bounds of an extended gadget. The gadget handler function (see FormGadgetHandlerType) should handle this event.

For this event, the data field contains the following structure:

```c
struct gadgetEnter {
    UInt16 gadgetID;
    struct FormGadgetType *gadgetP;
} gadgetEnter;
```

**Field Descriptions**

- **gadgetID**  
  Developer-defined ID of the gadget.

- **gadgetP**  
  Pointer to the FormGadgetType object representing this gadget.

**Compatibility**  
Implemented only if 3.5 New Feature Set is present.
frmGadgetMiscEvent
An application may choose to send this event when it needs to pass information to an extended gadget. The FrmHandleEvent function passes frmGadgetMiscEvents on to the extended gadget’s handler function (see FormGadgetHandlerType).
For this event, the data field contains the following structure:

```c
struct gadgetMisc {
    UInt16 gadgetID;
    struct FormGadgetType *gadgetP;
    UInt16 selector;
    void *dataP;
} gadgetMisc;
```

**Field Descriptions**

- **gadgetID**: Developer-defined ID of the gadget.
- **gadgetP**: Pointer to the `FormGadgetType` object representing this gadget.
- **selector**: Any necessary integer value to pass to the gadget handler function.
- **dataP**: A pointer to any necessary data to pass to the gadget handler function.

**Compatibility**

Implemented only if 3.5 New Feature Set is present.

**frmGotoEvent**

An application may choose to send itself this event when it receives a `sysAppLaunchCmdGoto` launch code. `sysAppLaunchCmdGoto` is generated when the user selects a record in the global find facility. Like `frmOpenEvent`, `frmGotoEvent` is a request that the application initialize and draw a form, but this event provides extra information so that the application may display and highlight the matching string in the form.

The application is responsible for handling this event.

For this event, the data field contains the following structure:

```c
struct frmGoto {
    UInt16 formID;
    UInt16 recordNum;
    UInt16 matchPos;
    UInt16 matchLen;
    UInt16 matchFieldNum;
```
Field Descriptions

**frmLoadEvent**

The form routines `FrmGotoForm` and `FrmPopupForm` send this event. It’s a request that the application load a form into memory.

The application is responsible for handling this event. In response to this event, applications typically initialize the form, make it active, and set the event handler.

For this event, the data field contains the following structure:

```c
struct frmLoad {
    UInt16 formID;
} frmLoad;
```

Field Descriptions

formID Developer-defined ID of the form.

**frmOpenEvent**

The form routines `FrmGotoForm` and `FrmPopupForm` send this event. It is a request that the application initialize and draw a form.

The application is responsible for handling this event.
For this event, the data field contains the following structure:

    struct frmOpen {
        UInt16 formID;
    } frmOpen;

Field Descriptions

formID    Developer-defined ID of the form.

**frmSaveEvent**

The form routine `FrmSaveAllForms` sends this event. It is a request that the application save any data stored in a form.

The application is responsible for handling this event.

No data is passed with this event.

**frmTitleEnterEvent**

The control routine `FrmHandleEvent` sends this event when it receives a `penDownEvent` within the bounds of the title of the form. Note that only the written title, not the whole title bar is active.

For this event, the data field contains the following structure:

    struct frmTitleEnter {
        UInt16 formID;
    } frmTitleEnter;

Field Descriptions

formID    Developer-defined ID of the form.

**frmTitleSelectEvent**

The control routine `FrmHandleEvent` sends this event. `FrmHandleEvent` receives a `frmTitleEnterEvent`, it tracks the pen until the pen is lifted. If the pen is lifted within the bounds of the active same title bar region, a `frmTitleSelectEvent` is added to the event queue.

For this event, the data field contains the following structure:
struct frmTitleSelect {
    UInt16 formID;
} frmTitleSelect;

Field Descriptions

formID Developer-defined ID of the form.

Compatibility

In Palm OS version 3.5 and higher, FrmHandleEvent responds to frmTitleSelectEvent. Its response is to enqueue a
keyDownEvent with a vchrMenu character to display the form’s menu.

frmUpdateEvent

The form routine FrmUpdateForm, or in some cases the routine FrmEraseForm, sends this event when it needs to redraw the
region obscured by the form being erased.

Generally, the region obscured by a form is saved and restored by
the form routines without application intervention. However, in
cases where the system is running low on memory, the form’s
routine may not save obscured regions itself. In that case, the
application adds a frmUpdateEvent to the event queue. The form
receives the event and redraws the region using the updateCode
value.

An application can define its own updateCode and then use this
event to also trigger behavior in another form, usually when
changes made to one form need to be reflected in another form.

For this event, the data field contains the following structure:

struct frmUpdate {
    UInt16 formID;
    UInt16 updateCode;
} frmUpdate;
Field Descriptions

formID  Developer-defined ID of the form.

updateCode  The reason for the update request. FrmEraseForm sets this code to frmRedrawUpdateCode, which indicates that the entire form needs to be redrawn. Application developers can define their own updateCode. The updateCode is passed as a parameter to FrmUpdateForm.

inetSockReadyEvent

This event is returned only by INetLibGetEvent (not EvtGetEvent) when the Internet library determines that a socket has data ready for an INetLibSockRead.

For this event, the data field contains the following structure:

```c
struct {
    MemHandle  sockH;
    UInt32     context;
    Boolean    inputReady;
    Boolean    outputReady;
} inetSockReady;
```

Field Descriptions

sockH  Socket handle of the socket that this event refers to.

context  Not used.

inputReady  true when the socket has data ready for the INetLibSockRead call.

outputReady  Not used.

The penDown, tapCount, screenX and screenY fields are not valid for Internet library events and should be ignored.

Compatibility  Implemented only if Wireless Internet Feature Set is present.
inetSockStatusChangeEvent

This event is returned only by INetLibGetEvent (not EvtGetEvent) when the Internet library determines that a socket has data ready for an INetLibSockRead.

For this event, the data field contains the following structure:

```
struct {
    MemHandle   sockH;
    UInt32      context;
    UInt16      status;
    Err         sockErr;
}inetSockStatusChange;
```

Field Descriptions

sockH

Socket handle of the socket that this event refers to.

current

Not used.

status

Current status of the socket. This is one of the INetStatusEnum constants.

sockErr

Reason for failure of the last operation, if any. The current socket error can be cleared by calling INetLibSockStatus.

The penDown, tapCount, screenX and screenY fields are not valid for Internet library events and should be ignored.

Compatibility

Implemented only if Wireless Internet Feature Set is present.

keyDownEvent

This event is sent by the system when the user enters a Graffiti® character, presses one of the buttons below the display, or taps one of the icons in the icon area; for example, the Find icon.
For this event, the data field contains the following structure:

```
struct _KeyDownEventType {
    WCHAR    chr;
    UINT16   keyCode;
    UINT16   modifiers;
};
```

**Field Descriptions**

- **chr** The character code.
- **keyCode** Unused.
- **modifiers** 0, or one or more of the following values:
  - **shiftKeyMask** Graffiti is in case-shift mode.
  - **capsLockMask** Graffiti is in cap-shift mode.
  - **numLockMask** Graffiti is in numeric-shift mode.
  - **commandKeyMask** The Graffiti glyph was the menu command glyph or a virtual key code.
  - **optionKeyMask** Not implemented. Reserved.
  - **controlKeyMask** Not implemented. Reserved.
  - **autoRepeatKeyMask** Event was generated due to auto-repeat.
  - **doubleTapKeyMask** Not implemented. Reserved.
  - **poweredOnKeyMask** The key press caused the system to be powered on.
  - **appEvtHookKeyMask** System use only.
  - **libEvtHookKeyMask** System use only.

**lstEnterEvent**

The list routine `LstHandleEvent` sends this event when it receives a `penDownEvent` within the bounds of a list object.

For this event, the data field contains the following structure:
Palm OS Events
Event Reference

```c
struct lstEnter {
    UInt16 listID;
    struct ListType *pList;
    Int16 selection;
} lstEnter;
```

**Field Descriptions**

- **listID**  Developer-defined ID of the list.
- **pList**  Pointer to a list structure (ListType).
- **selection**  Unused.

**lstExitEvent**

The list routine `LstHandleEvent` sends this event. When `LstHandleEvent` receives a `lstEnterEvent`, it tracks the pen until the pen is lifted. If the pen is lifted within the bounds of a list, a `lstSelectEvent` is added to the event queue; if not, a `lstExitEvent` is added to the event queue.

For this event, the data field contains the following structure:

```c
struct lstExit {
    UInt16 listID;
    struct ListType *pList;
} lstExit;
```

**Field Descriptions**

- **listID**  Developer-defined ID of the list.
- **pList**  Pointer to a list structure (ListType).

**lstSelectEvent**

The list routine `LstHandleEvent` sends this event. When `LstHandleEvent` receives a `lstEnterEvent`, it tracks the pen until the pen is lifted. If the pen is lifted within the bounds of a list, a `lstSelectEvent` is added to the event queue; if not, a `lstExitEvent` is added to the event queue.
Note that popup lists don’t generate a \lstSelectEvent. Instead, they generate a \popSelectEvent.

For this event, the data field contains the following structure:

```c
struct lstSelect {
    UInt16 listID;
    struct ListType *pList;
    Int16 selection;
} lstSelect;
```

**Field Descriptions**

- **listID**  
  Developer-defined ID of the list.
- **pList**  
  Pointer to a list structure (\ListType).
- **selection**  
  Item number (zero-based) of the new selection.

**menuCloseEvent**

This event is not currently used.

**menuCmdBarOpenEvent**

The menu routine \MenuHandleEvent sends this event when the user enters the menu shortcut keystroke, causing the command toolbar to be displayed at the bottom of the screen. Applications might respond to this event by calling \MenuCmdBarAddButton to add custom buttons to the command toolbar. Shared libraries or other non-application code resources can add buttons to the toolbar by registering to receive the \sysNotifyMenuCmdBarOpenEvent notification.

For this event, the data field contains the following structure:

```c
struct menuCmdBarOpen {
    Boolean preventFieldButtons;
    UInt8 reserved;
} menuCmdBarOpen;
```
Field Descriptions

preventFieldButtons If true, the field manager does not add the standard cut, copy, paste, and undo buttons when the focus is on a field. If false, the field adds the buttons.

reserved Unused.

To prevent the command toolbar from being displayed, respond to this event and return true. Returning true prevents the form manager from displaying the toolbar.

Compatibility Implemented only if 3.5 New Feature Set is present.

menuEvent

The menu routine MenuHandleEvent sends this event:

- When the user selects an item from a pull-down menu
- When the user selects a menu command using the Graffti command keystroke followed by an available command; for example, Command-C for copy
- When the user taps one of the buttons on the command toolbar and the button is set up to generate a menuEvent.

For this event, the data field contains the following structure:

```c
struct menu {
    UInt16 itemID;
} menu;
```

Field Descriptions

itemID Item ID of the selected menu command.

menuOpenEvent

The menu routine MenuHandleEvent sends this event when a new active menu has been initialized. A menu becomes active the first time the user taps the Menu silk-screen button or taps the form’s titlebar, and it might need to be re-initialized and reactivated several times during the life of an application.
A menu remains active until one of the following happens:

- A **FrmSetMenu** call changes the active menu on the form.
- A new form, even a modal form or alert panel, becomes active.

Suppose a user selects your application’s About item from the Options menu then clicks the OK button to return to the main form. When the About dialog is displayed, it becomes the active form, which causes the main form’s menu state to be erased. This menu state is not restored when the main form becomes active again. The next time the user requests the menu, it must be initialized again, so menuOpenEvent is sent again.

Applications might respond to this event by adding, hiding, or unhiding menu items using the functions **MenuAddItem**, **MenuHideItem**, or **MenuShowItem**.

A menuCloseEvent is defined by the system, but it is not currently sent. If you need to perform some cleanup (such as closing a resource) after the menu item you added is no longer needed, do so in response to **frmCloseEvent**.

For this event, the data field contains the following structure:

```c
struct menuOpen {
    UInt16    menuRscID;
    Int16     cause;
} menuOpen;
```

**Field Descriptions**

- **menuRscID**  Resource ID of the menu.
- **cause**  Reason for opening the menu. If **menuButtonCause**, the user tapped the Menu silkscreen button or tapped the form’s titlebar, and the menu is going to be displayed. If **menuCommandCause**, the user entered the command keystroke, so the menu is becoming active without being displayed.

**Compatibility**  Implemented only if **3.5 New Feature Set** is present.
nilEvent
A nilEvent is useful for animation, polling, and similar situations. The event manager sends this event when there are no events in the event queue. This can happen if the routine EvtGetEvent is passed a time-out value (a value other than evtWaitForever, -1). If EvtGetEvent is unable to return an event in the specified time, it returns a nilEvent. Different Palm OS versions and different devices can send nilEvents under different circumstances, so you might receive a nilEvent even before the timeout has expired.

penDownEvent
The event manager sends this event when the pen first touches the digitizer.
The following data is passed with the event:

Field Descriptions
penDown Always true.
tapCount The number of taps received at this location.
screenX Window-relative position of the pen in pixels (number of pixels from the left bound of the window).
screenY Window-relative position of the pen in pixels (number of pixels from the top left of the window).

penMoveEvent
The event manager sends this event when the pen is moved on the digitizer. Note that several kinds of UI objects, such as controls and lists, track the movement directly, and no penMoveEvent is generated.
The following data is passed with the event:

Field Descriptions
penDown Always true.
tapCount The number of taps received at this location.
penUpEvent

The event manager sends this event when the pen is lifted from the digitizer. Note that several kinds of UI objects, such as controls and lists, track the movement directly, and no penUpEvent is generated.

For this event, the data field contains the following structure:

```c
struct _PenUpEventType {
    PointType start;
    PointType end;
};
```

Field Descriptions

- **start**: Display-relative start point of the stroke.
- **end**: Display-relative end point of the stroke.

In addition, the following data is passed with this event:

- **penDown**: Always false.
- **tapCount**: The number of taps received at this location.
- **screenX**: Window-relative position of the pen in pixels (number of pixels from the left bound of the window).
- **screenY**: Window-relative position of the pen in pixels (number of pixels from the top left of the window).

popSelectEvent

The form routine FrmHandleEvent sends this event when the user selects an item in a popup list.

For this event, the data field contains the following structure:
struct popSelect {
    UInt16 controlID;
    struct ControlType *controlP;
    UInt16 listID;
    struct ListType *listP;
    Int16 selection;
    Int16 priorSelection;
} popSelect;

Field Descriptions

controlID          Developer-defined ID of the resource.
controlP           Pointer to the control structure
                   (ControlType) of the popup trigger object.
listID             Developer-defined ID of the popup list object.
listP              Pointer to the list structure (ListType) of the
                   popup list object.
selection          Item number (zero-based) of the new list
                   selection.
priorSelection     Item number (zero-based) of the prior list
                   selection.

sclEnterEvent

The routine SclHandleEvent sends this event when it receives a
penDownEvent within the bounds of a scroll bar.

Applications usually don’t have to handle this event.

For this event, the data field contains the following structure:

struct sclEnter {
    UInt16 scrollBarID;
    struct ScrollBarType *pScrollBar;
} sclEnter;
Field Descriptions

sclExitEvent

The routine SclHandleEvent sends this event when the user lifts the pen from the scroll bar.

Applications that want to implement non-dynamic scrolling should wait for this event, then scroll the text using the values provided in value and newValue.

Note that this event is sent regardless of previous sclRepeatEvents. If, however, the application has implemented dynamic scrolling, it doesn’t have to catch this event.

For this event, the data field contains the following structure:

```c
struct sclExit {
    UInt16 scrollBarID;
    struct ScrollBarType *pScrollBar;
    Int16 value;
    Int16 newValue;
} sclExit;
```

Field Descriptions

sclRepeatEvent

The routine SclHandleEvent sends this event when the pen is continually held within the bounds of a scroll bar.

```c
struct sclExit {
    UInt16 scrollBarID;
    struct ScrollBarType *pScrollBar;
    Int16 value;
    Int16 newValue;
} sclExit;
```
Applications that implement dynamic scrolling should watch for this event. In dynamic scrolling, the display is updated as the user drags the scroll bar (not after the user releases the scroll bar).

For this event, the data field contains the following structure:

```c
struct sclRepeat {
    Uint16 scrollBarID;
    struct ScrollBarType *pScrollBar;
    Int16 value;
    Int16 newValue;
    Int32 time;
} sclRepeat;
```

**Field Descriptions**

- **ScrollBarID**
  Developer-defined ID of the scroll bar resource.

- **pScrollBar**
  Pointer to the scroll bar structure.

- **value**
  Initial position of the scroll bar.

- **newValue**
  New position of the scroll bar. Given value and newValue, you can actually tell how much you have scrolled.

- **time**
  System-ticks count when the event is added to the queue to determine when the next event should occur.

**tblEnterEvent**

The table routine **TblHandleEvent** sends this event when it receives a **penDownEvent** within the bounds of an active item in a table object.

For this event, the data field contains the following structure:

```c
struct tblEnter {
    Uint16 tableID;
    struct TableType *pTable;
    Int16 row;
    Int16 column;
} tblEnter;
```
Field Descriptions

tableID  Developer-defined ID of the table.
pTable  Pointer to a table structure (TableType).
row  Row of the item.
column  Column of the item.

tblExitEvent

The table routine TblHandleEvent sends this event. When TblHandleEvent receives a tblEnterEvent, it tracks the pen until it's lifted from the display. If the pen is lifted within the bounds of the same item it went down in, a tblSelectEvent is added to the event queue; if not, a tblExitEvent is added to the event queue.

For this event, the data field contains the following structure:

```c
struct tblExit {
    UInt16 tableID;
    struct TableType *pTable;
    Int16 row;
    Int16 column;
} tblExit;
```

Field Descriptions

tableID  Developer-defined ID of the table.
pTable  Pointer to a table structure (TableType).
row  Row of the item.
column  Column of the item.

tblSelectEvent

The table routine TblHandleEvent sends this event. When TblHandleEvent receives a tblEnterEvent, it tracks the pen until the pen is lifted from the display. If the pen is lifted within the bounds of the same item it went down in, a tblSelectEvent is
added to the event queue; if not, a `tblExitEvent` is added to the event queue.

For this event, the data field contains the following structure:

```c
struct tblSelect {
    UInt16 tableID;
    struct TableType *pTable;
    Int16 row;
    Int16 column;
} tblSelect;
```

**Field Descriptions**

- **tableID**: Developer-defined ID of the table.
- **pTable**: Pointer to a table structure (`TableType`).
- **row**: Row of the item.
- **column**: Column of the item.

### `winEnterEvent`

The event manager sends this event when a window becomes the active window. This can happen in two ways: a call to `WinSetActiveWindow` is issued (`FrmSetActiveForm` calls this routine), or the user taps within the bounds of a window that is visible but not active. All forms are windows, but not all windows are forms; for example, the menu bar is a window but not a form.

For this event, the data field contains the following structure:

```c
struct _WinEnterEventType {
    WinHandle enterWindow;
    WinHandle exitWindow;
};
```
Field Descriptions

**enterWindow**  Handle to the window we are entering. If the window is a form, then this is a pointer to a `FormType` structure; if not, it’s a pointer to a `WindowType` structure.

**exitWindow**  Handle to the window we are exiting, if there is currently an active window, or zero if there is no active window. If the window is a form, then this is a pointer to a `FormType` structure; if not, it’s a pointer to a `WindowType` structure.

### winExitEvent

This event is sent by the event manager when a window is deactivated. A window is deactivated when another window becomes the active window (see `winEnterEvent`).

For this event, the data field contains the following structure:

```c
struct _WinExitEventType {  
    WinHandle enterWindow;  
    WinHandle exitWindow;  
};
```

Field Descriptions

**enterWindow**  Handle to the window we are entering. If the window is a form, then this is a pointer to a `FormType` structure; if not, it’s a pointer to a `WindowType` structure.

**exitWindow**  Handle to the window we are exiting. If the window is a form, then this is a pointer to a `FormType` structure; if not, it’s a pointer to a `WindowType` structure.
Notifications

This chapter provides detailed information about the notifications declared in the header file NotifyMgr.h. Notifications are broadcast to inform applications, shared libraries, system extensions, or other code resources of certain system-level or application-level events.

Notifications are similar to application launch codes, but they differ from launch codes in the following ways:

- The system broadcasts notifications only to interested parties. To register to receive a notification, use SysNotifyRegister.
- Notifications can be sent to non-applications.

See the “Notification Manager” chapter in this book and the section “Notifications” on page 30 of the Palm OS Programmer’s Companion, vol. I for more information on receiving and handling notifications.

Table 3.1 Notification Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cncNotifyProfileEvent</td>
<td>The connection profile used by the Connection Panel has changed.</td>
</tr>
<tr>
<td>sysExternalConnectorAttachEvent</td>
<td>A device has been attached to an external connector.</td>
</tr>
<tr>
<td>sysExternalConnectorDetachEvent</td>
<td>A device has been detached from an external connector.</td>
</tr>
<tr>
<td>sysNotifyAntennaRaisedEvent</td>
<td>The antenna has been raised on a Palm VII™ series device.</td>
</tr>
<tr>
<td>sysNotifyAppLaunchingEvent</td>
<td>An application is about to be launched.</td>
</tr>
<tr>
<td>sysNotifyAppQuittingEvent</td>
<td>An application has just quit.</td>
</tr>
</tbody>
</table>
### Table 3.1 Notification Constants (continued)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sysNotifyCardInsertedEvent</code></td>
<td>An expansion card has been inserted into the expansion slot.</td>
</tr>
<tr>
<td><code>sysNotifyCardRemovedEvent</code></td>
<td>An expansion card has been removed from the expansion slot.</td>
</tr>
<tr>
<td><code>sysNotifyDBCreatedEvent</code></td>
<td>A database has been created.</td>
</tr>
<tr>
<td><code>sysNotifyDBChangedEvent</code></td>
<td>Database info has been set on a database, such as with <code>DmSetDatabaseInfo</code>.</td>
</tr>
<tr>
<td><code>sysNotifyDBDeletedEvent</code></td>
<td>A database has been deleted.</td>
</tr>
<tr>
<td><code>sysNotifyDBDirtyEvent</code></td>
<td>A database has been opened for write or in some other way has been made modifiable.</td>
</tr>
<tr>
<td><code>sysNotifyDeleteProtectedEvent</code></td>
<td>The Launcher has attempted to delete a protected database.</td>
</tr>
<tr>
<td><code>sysNotifyDeviceUnlocked</code></td>
<td>The user has unlocked the device.</td>
</tr>
<tr>
<td><code>sysNotifyDisplayChangeEvent</code></td>
<td>The color table or bit depth has changed.</td>
</tr>
<tr>
<td><code>sysNotifyEarlyWakeupEvent</code></td>
<td>The system is starting to wake up.</td>
</tr>
<tr>
<td><code>sysNotifyEventDequeuedEvent</code></td>
<td>An event has been removed from the event queue with <code>EvtGetEvent</code>.</td>
</tr>
<tr>
<td><code>sysNotifyForgotPasswordEvent</code></td>
<td>The user has tapped the Lost Password button in the Security application.</td>
</tr>
<tr>
<td><code>sysNotifyGotUsersAttention</code></td>
<td>The Attention Manager has informed the user of an event.</td>
</tr>
<tr>
<td><code>sysNotifyHelperEvent</code></td>
<td>An application has requested that a particular service be performed.</td>
</tr>
<tr>
<td><code>sysNotifyIdleTimeEvent</code></td>
<td>The system is idle and is about to doze.</td>
</tr>
</tbody>
</table>
### Table 3.1 Notification Constants (continued)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sysNotifyInsPtEnableEvent</code></td>
<td>The insertion point is being enabled or disabled.</td>
</tr>
<tr>
<td><code>sysNotifyIrDASniffEvent</code></td>
<td>Not used.</td>
</tr>
<tr>
<td><code>sysNotifyKeyboardDialogEvent</code></td>
<td>The keyboard dialog is about to be displayed.</td>
</tr>
<tr>
<td><code>sysNotifyLateWakeupEvent</code></td>
<td>The system has finished waking up.</td>
</tr>
<tr>
<td><code>sysNotifyLocaleChangedEvent</code></td>
<td>The system locale has changed.</td>
</tr>
<tr>
<td><code>sysNotifyMenuCmdBarOpenEvent</code></td>
<td>The system is about to display the menu command toolbar.</td>
</tr>
<tr>
<td><code>sysNotifyNetLibIFMediaEvent</code></td>
<td>The system has been connected to or disconnected from the network.</td>
</tr>
<tr>
<td><code>sysNotifyPhoneEvent</code></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td><code>sysNotifyPOSEMountEvent</code></td>
<td>System use only.</td>
</tr>
<tr>
<td><code>sysNotifyProcessPenStrokeEvent</code></td>
<td>The user has made a pen stroke on the silkscreen portion of the digitizer.</td>
</tr>
<tr>
<td><code>sysNotifyResetFinishedEvent</code></td>
<td>The system has finished a reset.</td>
</tr>
<tr>
<td><code>sysNotifyRetryEnqueueKey</code></td>
<td>The Attention Manager has failed to post a virtual character to the key queue.</td>
</tr>
<tr>
<td><code>sysNotifySleepNotifyEvent</code></td>
<td>The system is about to go to sleep.</td>
</tr>
<tr>
<td><code>sysNotifySleepRequestEvent</code></td>
<td>The system has decided to go to sleep.</td>
</tr>
<tr>
<td><code>sysNotifySyncFinishEvent</code></td>
<td>A HotSync® operation has just completed.</td>
</tr>
<tr>
<td><code>sysNotifySyncStartEvent</code></td>
<td>A HotSync operation is about to begin.</td>
</tr>
<tr>
<td><code>sysNotifyTimeChangeEvent</code></td>
<td>The system time has just changed.</td>
</tr>
<tr>
<td><code>sysNotifyVirtualCharHandlingEvent</code></td>
<td>A virtual character is being handled.</td>
</tr>
</tbody>
</table>
Notification Data Structures

SysNotifyParamType

The `SysNotifyParamType` structure contains all of the data associated with a notification. This structure is passed as the parameter block for the `sysAppLaunchCmdNotify` launch code or as a parameter to the notification callback function. All notifications have some common data. Most notifications also have data specific to that notification. The specific data is pointed to by the `notifyDetailsP` field.

The common data for each notification is documented below the following structure declaration. The Notification Reference section gives details on the important data associated with each type of notification.

```c
typedef struct SysNotifyParamType {
    UInt32   notifyType;
    UInt32   broadcaster;
    void *   notifyDetailsP;
    void *   userDataP;
    Boolean  handled;
    UInt8    reserved2;
} SysNotifyParamType;
```

Table 3.1 Notification Constants (continued)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sysNotifyVolumeMountedEvent</code></td>
<td>A file system has been mounted.</td>
</tr>
<tr>
<td><code>sysNotifyVolumeUnmountedEvent</code></td>
<td>A file system has been unmounted.</td>
</tr>
</tbody>
</table>

Notification Data Structures
Field Descriptions

notifyType  The type of event that occurred. See Notification Reference.

broadcaster  The creator ID of the application that broadcast the notification, or sysNotifyBroadcasterCode if the system broadcast the event.

notifyDetailsP  Pointer to data specific to this notification.

userDataP  Custom data that your notification handler requires. You create this data and pass it to SysNotifyRegister.

handled  Set to true if the notification has been handled; set to false otherwise. In some cases, handled is treated as a bit field that notification handlers can use to indicate that certain conditions are true.

reserved2  Reserved for future use.

Notification Reference

cncNotifyProfileEvent

The cncNotifyProfileEvent is broadcast whenever a connection profile has been created, modified, or deleted and after a request has been made to update the connection profile list.

The notifyDetailsP field informs the notification handler of the type of change that was made. Register for the cncNotifyProfileEvent if your application maintains its own list of connection profiles that it should keep current or if it should help the Connection Panel maintain its list.

cncNotifyProfileEvent Specific Data

notifyDetailsP points to a CncProfileNotifyDetailsType structure.
typedef struct _CncProfileNotifyDetailsTag {
    UInt16 version;
    UInt32 profileID;
    UInt16 deviceKind;
    UInt16 request;
} CncProfileNotifyDetailsType;

**Fields**

- **version**
  
  The current version of this structure. Use the `kCncProfileNotifyCurrentVersion` constant to find out what the current version is.

- **profileID**
  
  The ID of the modified connection profile.

- **deviceKind**
  
  Device kind of the profile. This can be one of the following constants:
  
  - `kCncDeviceKindSerial`
    
    Serial connection profile.
  
  - `kCncDeviceKindModem`
    
    Modem profile.
  
  - `kCncDeviceKindPhone`
    
    Phone profile.
  
  - `kCncDeviceKindLocalNetwork`
    
    LAN profile.

- **request**
  
  The action that was performed. This can be one of the following constants:
  
  - `kCncNotifyCreateRequest`
    
    The profile has been created.
  
  - `kCncNotifyDeleteRequest`
    
    The profile is about to be deleted.
  
  - `kCncNotifyModifyRequest`
    
    The profile has been modified.
kCncNotifyUpdateListRequest
A HotSync operation or system reset has just occurred. The notification handler should update the Connection Panel’s list.

If a profile has been created or modified, the request field also contains a flag indicating how the current profile is to be set:

kCncBecomeCurrentModifier
The new profile should be made the current profile.

kCncNotifyAlertUserModifier
The user is prompted to set the current profile.

Compatibility
Implemented only if 4.0 New Feature Set is present.

sysExternalConnectorAttachEvent
The sysExternalConnectorAttachEvent is broadcast when a USB cradle, RS-232 cradle or peripheral, a power cable, or a modem is attached to the universal connector. This notification is broadcast only on devices that have the universal connector.

sysExternalConnectorAttachEvent Specific Data
The notifyDetailsP field points to a UInt16 that identifies which type of device was attached.

Compatibility
Implemented only if 4.0 New Feature Set is present.

sysExternalConnectorDetachEvent
The sysExternalConnectorDetachEvent is broadcast when a USB cradle, a RS-232 cradle or peripheral, a power cable, or a modem is detached from the universal connector. This notification is only broadcast on devices that have the universal connector.
sysExternalConnectorDetachEvent Specific Data
The notifyDetailsP field points to a UInt16 that identifies which type of device was detached.

Compatibility Implemented only if 4.0 New Feature Set is present.

sysNotifyAntennaRaisedEvent
The sysNotifyAntennaRaisedEvent is broadcast by SysHandleEvent when the antenna is raised on a Palm VII series device.

Register for this notification if you want to handle the antenna key down event. To ensure that no other code handles the antenna key down event after yours, set the handled parameter of the SysNotifyParamType structure to true.

sysNotifyAntennaRaisedEvent Specific Data
None.

Compatibility Implemented only if Notification Feature Set is present.

sysNotifyAppLaunchingEvent
The sysNotifyAppLaunchingEvent is broadcast before an application is launched with sysAppLaunchCmdNormalLaunch.

sysNotifyAppLaunchingEvent Specific Data
notifyDetailsP points to a SysNotifyAppLaunchOrQuitType structure.

Prototype
typedef struct SysNotifyAppLaunchOrQuitTag {
  UInt32    version;
  UInt32    dbID;
  UInt16    cardNo;
} SysNotifyAppLaunchOrQuitType;

Fields version The current version of this structure. The current version is 0.
dbID                  The local ID of the application.
cardNo                The number of the card on which the application resides.

Compatibility        This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don’t broadcast this notification. Palm OS 5 does broadcast it. Later versions may or may not broadcast this notification.

sysNotifyAppQuittingEvent
The sysNotifyAppQuittingEvent is broadcast right after an application that was launched with sysAppLaunchCmdNormalLaunch quits.

sysNotifyAppLaunchingEvent Specific Data
notifyDetailsP points to a SysNotifyAppLaunchOrQuitType structure. See the description of sysNotifyAppLaunchingEvent for a description of this structure.

Compatibility        This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don’t broadcast this notification. Palm OS 5 does broadcast it. Later versions may or may not broadcast this notification.

sysNotifyCardInsertedEvent
The sysNotifyCardInsertedEvent is broadcast when an Expansion Manager card is inserted into a slot. When a new card is inserted, the Expansion Manager attempts to mount the volume on that card and plays a sound (indicating success or failure) once the attempt is complete.

Most applications will want to register for sysNotifyVolumeMountedEvent instead of this notification. Register for sysNotifyCardInsertedEvent if you need to know when a card is inserted or if you want to prevent the Expansion Manager from performing its default handling of the notification.
To prevent the Expansion Manager from mounting the volume, set the \texttt{expHandledVolume} bit in the \texttt{handled} field. To prevent the Expansion Manager from playing the sound, set the \texttt{expHandledSound} bit in the \texttt{handled} field. For example:

\begin{verbatim}
    cmdPBP->handled |= expHandledSound;
\end{verbatim}

\textbf{sysNotifyCardInsertedEvent Specific Data}
\begin{itemize}
    \item \texttt{notifyDetailsP} points to a \texttt{UInt16} containing the slot reference number.
\end{itemize}

\textbf{Compatibility}
Implemented only if 4.0 New Feature Set is present.

\textbf{sysNotifyCardRemovedEvent}
The \texttt{sysNotifyCardRemovedEvent} is broadcast when an Expansion Manager card is removed from a slot. When a card is removed, the Expansion Manager responds to this notification by playing a goodbye sound and then attempting to unmount the volume.

Most applications will want to register for \texttt{sysNotifyVolumeUnmountedEvent} instead of this notification. Register for \texttt{sysNotifyCardRemovedEvent} if you need to know when a card is removed or if you want to prevent the Expansion Manager from performing its default handling of the notification.

To prevent the Expansion Manager from unmounting the volume, set the \texttt{expHandledVolume} bit in the \texttt{handled} field. To prevent the Expansion Manager from playing the sound, set the \texttt{expHandledSound} bit in the \texttt{handled} field. For example:

\begin{verbatim}
    cmdPBP->handled |= expHandledSound;
\end{verbatim}

\textbf{sysNotifyCardRemovedEvent Specific Data}
\begin{itemize}
    \item \texttt{notifyDetailsP} points to a \texttt{UInt16} containing the slot reference number.
\end{itemize}

\textbf{Compatibility}
Implemented only if 4.0 New Feature Set is present.
New

sysNotifyDBCreatedEvent

The sysNotifyDBCreatedEvent is broadcast sometime after a database is created with DmCreateDatabase.

Register for this notification if you keep an internal list of databases that needs to be updated when a new database is created.

**IMPORTANT:** The sysNotifyDBxxxEvent notifications are deferred notifications. So, for instance, if your application creates a database, opens it for write, and then renames it, all before EvtGetEvent is called, the three corresponding notifications will all go out together. A sysNotifyDBDirtyEvent handler would fail if it tried to open the database, since the database will already have been renamed. You must be aware of the ramifications of a deferred notification when writing your notification handler.

sysNotifyDBCreatedEvent Specific Data

`notifyDetailsP` points to a SysNotifyDBCreatedType structure.

Prototype

```c
typedef struct SysNotifyDBCreatedTag {
    Char dbName[dmDBNameLength];
    UInt32 creator;
    UInt32 type;
    LocalID newDBID;
    UInt16 cardNo;
    Boolean resDB;
    UInt8 padding;
} SysNotifyDBCreatedType;
```

Fields

- **dbName**: Database name.
- **creator**: Database creator ID.
- **type**: Database type.
- **newDBID**: Local ID of the newly-created database.
cardno Card number upon which the database resides.
resDB true if the database is a resource database, false otherwise.
padding Structure padding byte.

Compatibility Implemented only if 5.0 New Feature Set is present.

New **sysNotifyDBChangedEvent**

The **sysNotifyDBChangedEvent** is broadcast sometime *after* database info is set with **DmSetDatabaseInfo**.

Register for this notification if you keep an internal list of databases that needs to be updated when database info changes.

**IMPORTANT:** The **sysNotifyDB*xxxEvent** notifications are deferred notifications. So, for instance, if your application creates a database, opens it for write, and then renames it, all before **EvtGetEvent** is called, the three corresponding notifications will all go out together. A **sysNotifyDBDirtyEvent** handler would fail if it tried to open the database, since the database will already have been renamed. You must be aware of the ramifications of a deferred notification when writing your notification handler.

**sysNotifyDBChangeEvent Specific Data**

**notifyDetailsP** points to a **SysNotifyDBChangedType** structure. The contents of fields in this structure indicates what about the database changed, and thus which of the other structure fields contain valid data.
Notifications
Notification Reference

Prototype
typedef struct SysNotifyDBChangedTag {
    Char dbName[dmDBNameLength];
    LocalID dbID;
    UInt32 creator;
    UInt32 type;
    UInt32 crDate;
    UInt32 modDate;
    UInt32 bckUpDate;
    UInt32 modNum;
    LocalID appInfoID;
    LocalID sortInfoID;
    UInt16 attributes;
    UInt16 cardNo;
    UInt16 version;
    UInt16 fields;
    Char oldName[dmDBNameLength];
    UInt32 oldCreator;
    UInt32 oldType;
    UInt16 oldAttributes;
    UInt16 padding;
} SysNotifyDBChangedType;

Fields
dbName       New name of database.
dbID         Database ID.
creator      New database creator ID.
type         New database type.
crDate       New database creation date.
modDate      New database modification date.
bckUpDate    New database backup date.
modNum       New database modification number.
appInfoID    New database application info block.
sortInfoID   New database sort info block.
attributes   New database attributes.
cardNo       Card number upon which the database resides.
Notifications

version
New database version.

fields
Flags that indicate what about the database changed, and thus which of the above fields are set. The constants that define the fields bits are:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBChangedFieldSetName</td>
<td>0x1</td>
</tr>
<tr>
<td>DBChangedFieldSetCreator</td>
<td>0x2</td>
</tr>
<tr>
<td>DBChangedFieldSetType</td>
<td>0x4</td>
</tr>
<tr>
<td>DBChangedFieldSetCrDate</td>
<td>0x8</td>
</tr>
<tr>
<td>DBChangedFieldSetModDate</td>
<td>0x10</td>
</tr>
<tr>
<td>DBChangedFieldSetBckUpDate</td>
<td>0x20</td>
</tr>
<tr>
<td>DBChangedFieldSetModNum</td>
<td>0x40</td>
</tr>
<tr>
<td>DBChangedFieldSetAppInfo</td>
<td>0x80</td>
</tr>
<tr>
<td>DBChangedFieldSetSortInfo</td>
<td>0x100</td>
</tr>
<tr>
<td>DBChangedFieldSetAttributes</td>
<td>0x200</td>
</tr>
<tr>
<td>DBChangedFieldSetVersion</td>
<td>0x400</td>
</tr>
</tbody>
</table>

oldName
Name of database prior to the call to DmSetDatabaseInfo.

oldCreator
Database creator ID prior to the call to DmSetDatabaseInfo.

oldType
Database type prior to the call to DmSetDatabaseInfo.

oldAttributes
Database attributes prior to the call to DmSetDatabaseInfo.

padding
Structure padding bytes.

Compatibility
Implemented only if 5.0 New Feature Set is present.
sysNotifyDBDeletedEvent

The sysNotifyDBDeletedEvent is broadcast sometime after a database is removed from the device.

Register for this notification if you keep an internal list of databases that needs to be updated upon removal of a database. For example, the Attention Manager and Connection Manager register for this notification to maintain their internal lists of databases.

**IMPORTANT:** The sysNotifyDBxxxEvent notifications are deferred notifications. So, for instance, if your application creates a database, opens it for write, and then renames it, all before EvtGetEvent is called, the three corresponding notifications will all go out together. A sysNotifyDBDirtyEvent handler would fail if it tried to open the database, since the database will already have been renamed. You must be aware of the ramifications of a deferred notification when writing your notification handler.

sysNotifyDBDeletedEvent Specific Data

notifyDetailsP points to a SysNotifyDBDeletedType structure.

**Prototype**

typedef struct SysNotifyDBDeletedTag {
    LocalID oldDBID;
    UInt16 cardNo;
    UInt16 attributes;
    Char dbName[dmDBNameLength];
    UInt32 creator;
    UInt32 type;
} SysNotifyDBDeletedType;

**Fields**

| oldDBID | The local ID of the deleted database. This ID is no longer valid. |

**WARNING!** The ID in oldDBID is invalid by the time the notification is broadcast. If you try to pass it to a Data Manager function, the system will crash.
cardNo The number of the card on which the database resided.

attributes The deleted database’s attributes.

dbName The name of the deleted database.

creator The creator ID of the deleted database.

type The type of the deleted database.

Compatibility Implemented only if 4.0 New Feature Set is present.

New sysNotifyDBDirtyEvent

The sysNotifyDBDirtyEvent is broadcast sometime after a database is opened for write or in some other way has been made modifiable. Note that the database may not have actually been modified yet.

Register for this notification if you keep an internal list of databases that needs to be updated when a database becomes “dirty.” For instance, upon reset the Launcher normally checks over such databases and updates its internal list.

IMPORTANT: The sysNotifyDBxxxEvent notifications are deferred notifications. So, for instance, if your application creates a database, opens it for write, and then renames it, all before EvtGetEvent is called, the three corresponding notifications will all go out together. A sysNotifyDBDirtyEvent handler would fail if it tried to open the database, since the database will already have been renamed. You must be aware of the ramifications of a deferred notification when writing your notification handler.

sysNotifyDBDirtyEvent Specific Data

notifyDetailsP points to a SysNotifyDBDirtyType structure.
Prototype

typedef struct SysNotifyDBDirtyTag {
    Char dbName[dmDBNameLength];
    UInt32 creator;
    UInt32 type;
} SysNotifyDBDirtyType;

Fields

dbName  Database name.
creator  Database creator ID.
type  Database type.

Compatibility  Implemented only if 5.0 New Feature Set is present.

sysNotifyDeleteProtectedEvent

The sysNotifyDeleteProtectedEvent is broadcast when the Launcher attempts to delete a database that has the protected flag set. The Launcher broadcasts the notification and then attempts to delete the database again. Any third party application that deletes databases should broadcast this notification as well.

Register for this notification if you have a protected database but you still want to allow users to delete your application or other code resource if they choose. A notification handler should check the information in the notifyDetailsP struct to see if its database is the one being deleted. If so, it should respond to this notification to perform any necessary cleanup and to clear the protected flag. In this way, when the Launcher attempts to delete the database again, it will succeed. Note that if an application has multiple protected databases, this notification may be sent out more than once.

sysNotifyDeleteProtectedEvent Specific Data

notifyDetailsP points to a SysNotifyDBInfoType structure.

Prototype

typedef struct SysNotifyDBInfoTag {
    LocalID dbID;
    UInt16 cardNo;
    UInt16 attributes;
    Char dbName[dmDBNameLength];
    UInt32 creator;
}
Notifications
Notification Reference

```c
UInt32 type;
} SysNotifyDBInfoType;
```

**Fields**
- **dbID** The local ID of the database to be deleted.
- **cardNo** The number of the card on which the database resides.
- **attributes** The database’s attributes.
- **dbName** The name of the database to be deleted.
- **creator** The creator ID of the database to be deleted.
- **type** The type of the database to be deleted.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

### sysNotifyDeviceUnlocked

The `sysNotifyDeviceUnlocked` notification is broadcast by the Security application when the user unlocks the device. The notification is broadcast immediately after the device has finished unlocking.

If you display UI in response to the `sysNotifyLateWakeupEvent` notification, you should also register to receive the `sysNotifyDeviceUnlocked` notification. When a locked device receives the `sysNotifyLateWakeupEvent`, your UI should not be displayed if the device is waiting for the user to enter the password. The `sysNotifyDeviceUnlocked` notification is broadcast after the password is entered, which indicates that the user interface is ready.

**sysNotifyDeviceUnlocked Specific Data**

None.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

### sysNotifyDisplayChangeEvent

The `sysNotifyDisplayChangeEvent` is broadcast whenever the display mode changes. That is, either the color table has been set to
use a specific palette using the WinPalette function or the bit
depth has changed using the WinScreenMode function.

The notifyDetailsP field indicates how the bit depth changed. If
the two values in the struct are equal, it means that the color palette
has changed instead of the bit depth.

sysNotifyDisplayChangeEvent Specific Data

notifyDetailsP points to a
SysNotifyDisplayChangeDetailsType structure.

Prototype
typedef struct {
    UInt32 oldDepth;
    UInt32 newDepth;
} SysNotifyDisplayChangeDetailsType;

Fields
oldDepth The old bit depth.
newDepth The new bit depth.

Compatibility Implemented only if Notification Feature Set is present.

sysNotifyEarlyWakeupEvent

The sysNotifyEarlyWakeupEvent is broadcast during
SysHandleEvent immediately after the system has finished
sleeping. The screen may still be turned off, and the system may not
fully wake up. It may simply handle an alarm or a battery charger
event and go back to sleep. Most applications that need notification
of a wakeup event will probably want to register for
sysNotifyLateWakeupEvent instead.

IMPORTANT: This notification is not guaranteed to be
broadcast. Thus, it is not suitable for applications where external
hardware must be turned on when the system is powered on.

sysNotifyEarlyWakeupEvent Specific Data

None.
**Compatibility**

Implemented only if Notification Feature Set is present.

**sysNotifyEventDequeuedEvent**

The `sysNotifyEventDequeuedEvent` is broadcast for each event removed from the event queue with `EvtGetEvent`.

**WARNING!** Be very careful about registering for this notification; it can result in significantly degraded system performance.

**sysNotifyEventDequeuedEvent Specific Data**

`notifyDetailsP` points to the dequeued event’s `EventType` structure.

**IMPORTANT:** For speed, the event structure that `notifyDetailsP` points to uses system-native endianness. This means that you might need to byte-swap the structure’s contents, depending on the endianness of the underlying operating system.

**Compatibility**

This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don’t broadcast this notification. Palm OS 5 does broadcast it. Later versions may or may not broadcast this notification.

**sysNotifyForgotPasswordEvent**

The `sysNotifyForgotPasswordEvent` is broadcast after the user taps the Lost Password button in the Security application. The notification is sent after the user has confirmed that all private records should be deleted but before the deletion actually occurs.

**sysNotifyForgotPasswordEvent Specific Data**

None.

**Compatibility**

Implemented only if Notification Feature Set is present.
sysNotifyGotUsersAttention

The `sysNotifyGotUsersAttention` notification is broadcast when the Attention Manager has finished displaying or sounding its attention indicators (blinking, playing sounds, vibrating, and so on).

System extensions or shared libraries should register for this notification if they want to perform some extra effect or if they simply want to be informed of when the user’s attention was received.

**sysNotifyGotUsersAttention Specific Data**

`notifyDetailsP` points to an `AttnNotifyDetailsType` structure.

**Prototype**

```c
typedef struct {
    AttnFlagsType flags;
} AttnNotifyDetailsType;
```

**Fields**

- **flags**
  - The attention indicators that were used to get the user’s attention. See `AttnFlagsType`.

**Compatibility**

Implemented only if [4.0 New Feature Set](#) is present.

sysNotifyHelperEvent

The `sysNotifyHelperEvent` is broadcast by applications to request a service from another application. For example, the Address Book application broadcasts this notification to request that the Dial application dial a phone number. For the `sysNotifyHelperEvent`, the notification client (that is, the application or shared library that registers for the notification) is called a **helper**.

The application that broadcasts this notification specifies one of the action codes listed in Table 35.1 in Chapter 35, “Helper API.” These action codes request all helper applications to enumerate (list the services they perform), validate (ensure that the service will succeed), and execute (perform the action). The helper responds to the notification by returning the required data in the appropriate
portion of the notifyDetailsP structure and by setting the handled field to true or false to indicate the success or failure of the action.

For more information on this notification, see the section “Helper Notifications” on page 38 in the Palm OS Programmer’s Companion, vol. I.

**sysNotifyHelperEvent Specific Data**

notifyDetailsP points to a HelperNotifyEventType structure.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.

**sysNotifyIdleTimeEvent**

The sysNotifyIdleTimeEvent is broadcast when the system is idle and is about to doze.

**Compatibility**

This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don’t broadcast this notification. Palm OS 5 does broadcast it. Later versions may or may not broadcast this notification.

**sysNotifyInsPtEnableEvent**

The sysNotifyInsPtEnableEvent is broadcast at the start of InsPtEnable.

**sysNotifyInsPtEnableEvent Specific Data**

notifyDetailsP points to a Boolean: the enableIt parameter passed to InsPtEnable.

**Compatibility**

This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don’t broadcast this notification. Palm OS 5 does broadcast it. Later versions may or may not broadcast this notification.
**sysNotifyKeyboardDialogEvent**

The `sysNotifyKeyboardDialogEvent` is broadcast whenever the system keyboard is displayed. It is intended to enable the replacement of `SysKeyboardDialog` function’s user interface.

**sysNotifyKeyboardDialogEvent Specific Data**

`notifyDetailsP` points to the `KeyboardType` enum that indicates the mode in which the keyboard should be opened: alphabetic, numeric, or international.

**Compatibility**

This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don’t broadcast this notification. Palm OS 5 does broadcast it. Later versions may or may not broadcast this notification.

**sysNotifyLateWakeupEvent**

The `sysNotifyLateWakeupEvent` is broadcast during `SysHandleEvent` immediately after the device has finished waking up. This notification is sent at the late stage of wakeup, after the screen has been turned on. When this notification is broadcast, the system is guaranteed to fully wake up. Register for this notification if you need to perform startup tasks each time the system wakes up.

**IMPORTANT:** This notification is **not** guaranteed to be broadcast. Thus, it is unsuitable for applications where external hardware must be powered on when the device wakes up.

When the device receives this notification, it may be locked and waiting for the user to enter the password. If this is the case, you must wait for the user to unlock the device before you display a user interface. Therefore, if you intend to display a user interface when the device wakes up, you should make sure the device is not locked. If the device is locked, you should register for `sysNotifyDeviceUnlocked` notification and display your user interface when it is received. For example:
case sysNotifyLateWakeupEvent:
    if ((Boolean)
        PrefGetPreference(prefDeviceLocked)) {
        SysNotifyRegister(myCardNo, myDbID,
            sysNotifyDeviceUnlocked, NULL,
            sysNotifyNormalPriority, NULL);
    } else {
        HandleDeviceWakeup();
    }

    case sysNotifyDeviceUnlocked:
        HandleDeviceWakeup();

Note that the sysNotifyDeviceUnlocked notification is only
broadcast on Palm OS 4.0 and higher.

**sysNotifyLateWakeupEvent Specific Data**

None.

**Compatibility**

Implemented only if Notification Feature Set is present.

**sysNotifyLocaleChangedEvent**

The sysNotifyLocaleChangedEvent is broadcast immediately
after the system locale has changed. Currently, the user has the
opportunity to change the locale only when the device first starts up
and after a hard reset.

RAM-based applications and other code resources should obtain
locale information by passing the prefLocale constant to
PrefGetPreference. They should not register for this
notification. This notification is used by the built-in applications,
which respond to it by rebuilding their default databases to use the
newly selected language and character set.

**sysNotifyLocaleChangedEvent Specific Data**

notifyDetailsP points to a SysNotifyLocaleChangedType
structure.

**Prototype**

typedef struct SysNotifyLocaleChangedTag {
    LmLocaleType oldLocale;
    LmLocaleType newLocale;
}
Notifications
Notification Reference

} SysNotifyLocaleChangedType;

**Fields**

- `oldLocale`  The old locale. See [LmLocaleType](#).
- `newLocale`  The new locale.

**Compatibility**

Implemented only if [4.0 New Feature Set](#) is present.

### sysNotifyMenuCmdBarOpenEvent

The `sysNotifyMenuCmdBarOpenEvent` is broadcast during `MenuHandleEvent` when it is about to display the menu shortcut command bar.

Register for this notification if you are writing a system extension (such as a “hack” installed with the HackMaster program) that needs to add a button to the menu command bar or to suppress the menu command bar. To add a button, call `MenuCmdBarAddButton`. To suppress the command toolbar, set the `handled` field to `true`.

Applications that need to add their own buttons to the menu command bar should do so in response to a `menuCmdBarOpenEvent`. They should not register for this notification because an application should only add buttons if it is already the active application. The notification is sent after the event has been received, immediately before the command toolbar is displayed.

### sysNotifyMenuCmdBarOpenEvent

None.

**Compatibility**

Implemented only if [Notification Feature Set](#) is present.

### sysNotifyNetLibIFMediaEvent

The `sysNotifyNetLibIFMediaEvent` is broadcast at the top of the event loop whenever the network interface makes the network connection active or inactive. The Network Panel uses this notification to decide whether the Connect button should be active.
Register for this notification if you need to know when the network connection is currently active.

**sysNotifyNetLibIFMediaEvent Specific Data**

`notifyDetailsP` contains a `SysNotifyNetLibIFMediaType` structure.

**Prototype**

```c
typedef struct SysNotifyNetLibIFMediaType {
    NetLibIFMediaEventNotificationTypeEnum eType;
    UInt32 ifCreator;
    UInt16 ifInstance;
} SysNotifyNetLibIFMediaType;
```

**Fields**

- **eType**
  
  One of the following values:
  - `netIFMediaUp`
    
    The network connection is active. This is usually sent after the network interface has displayed UI indicating that a connection attempt is in progress.
  - `netIFMediaDown`
    
    The network connection is inactive. This is usually sent after the network interface has brought the connection down because an inactivity timeout value was reached.

- **ifCreator**
  
  Creator ID of the network interface

- **ifInstance**
  
  Instance number of the network interface.

**Compatibility**

Implemented only if [4.0 New Feature Set](#) is present.

**sysNotifyProcessPenStrokeEvent**

The `sysNotifyProcessPenStrokeEvent` is broadcast to enable custom recognition of strokes made on the silkscreen portion of the digitizer.
sysNotifyProcessPenStrokeEvent Specific Data

notifyDetailsP points to a SysNotifyPenStrokeType structure.

Prototype
typedef struct SysNotifyPenStrokeTag {
    UInt32     version;
    PointType  startPt;
    PointType  endPt;
} SysNotifyPenStrokeType;

Fields
version
The current version of this structure. The current version is 0.
startPt
Start point of stroke.
endPt
End point of stroke.

Compatibility
This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don’t broadcast this notification. Palm OS 5 does broadcast it. Later versions may or may not broadcast this notification.

sysNotifyResetFinishedEvent

The sysNotifyResetFinishedEvent is broadcast immediately after the system has finished a reset.

Because the notification registry is cleared upon a reset, only internal system components use this notification. Applications that need to be informed of a system reset can respond to the sysAppLaunchCmdSystemReset launch code.

sysNotifyResetFinishedEvent Specific Data
None.

Compatibility
Implemented only if Notification Feature Set is present.

sysNotifyRetryEnqueueKey

The sysNotifyRetryEnqueueKey notification is broadcast at the top of the event loop if the Attention Manager has attempted to post
a virtual character to the key queue and failed because the queue is full. The notification signals that the Attention Manager is going to retry enqueuing the virtual character until it is successful.

Most applications do not need to register for this notification. It is used only by the Attention Manager to schedule retries of enqueuing the virtual character. When enqueuing a virtual character fails, the Attention Manager retries at the top of the event loop. It uses this notification to schedule retries so that they occur even if the user switches applications.

**sysNotifyRetryEnqueueKey Specific Data**

`notifyDetailsP` points to a `WChar` containing the virtual character to be enqueued.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.

**sysNotifySleepNotifyEvent**

The `sysNotifySleepNotifyEvent` is broadcast during `SysHandleEvent` immediately before the system is put to sleep. After the broadcast is complete, the system is put to sleep.

Register for this notification if you have a small amount of cleanup that needs to be performed before the system goes to sleep. It is recommended that you not perform any sort of prolonged activity, such as displaying an alert panel that requests confirmation, in response to a sleep notification. If you do, the alert might be displayed long enough to trigger another auto-off event, which could be detrimental to other handlers of this notification.

If your code is in the middle of a lengthy computation and needs to defer sleep, it should register for the `sysNotifySleepRequestEvent` instead.
**Important:** This notification is not guaranteed to be broadcast. For example, if the system goes to sleep because the user removes the batteries, sleep notifications are not sent. Thus, these notifications are unsuitable for applications where external hardware must be shut off to conserve power before the system goes to sleep.

**SysNotifySleepNotifyEvent** Specific Data

None.

**Compatibility**

Implemented only if [Notification Feature Set](#) is present.

**SysNotifySleepRequestEvent**

The `sysNotifySleepRequestEvent` is broadcast during `SysHandleEvent` processing when the system has decided to go to sleep.

Register for this notification if you need to delay the system from going to sleep while your code performs a lengthy operation, such as disconnecting from the network. The system checks the `deferSleep` value when each notification handler returns. If it is nonzero, it cancels the sleep event.

After you defer sleep, your code is free to finish what it was doing. When it is finished, you must allow the system to continue with the sleep event. To do so, create a `keydownEvent` with the `resumeSleepChr` and the command key bit set (to signal that the character is virtual) and add it to the event queue. When the system receives this event, it will again broadcast the `sysNotifySleepRequestEvent` to all clients. If `deferSleep` is 0 after all clients return, then the system knows it is safe to go to sleep, and it broadcasts the `sysNotifySleepNotifyEvent` to all of its clients.

Note that you may receive this notification several times before the system goes to sleep because notification handlers can delay the system sleep and resume it later.
IMPORTANT: This notification is not guaranteed to be broadcast. For example, if the system goes to sleep because the user removes the batteries, sleep notifications are not sent. Thus, these notifications are unsuitable for applications where external hardware must be shut off to conserve power before the system goes to sleep.

sysNotifySleepRequestEvent Specific Data

notifyDetailsP points to a SleepEventParamType structure.

Prototype

typedef struct {
    UInt16 reason;
    UInt16 deferSleep;
} SleepEventParamType;

Fields

reason The reason the system is going to sleep. The possible values are:

sysSleepAutoOff
    The idle time limit has been reached.

sysSleepPowerButton
    The user pressed the power off button.

sysSleepResumed
    The sleep event was deferred by one of the notification handlers but has been resumed through the use of the resumeSleepChr.

sysSleepUnknown
    Unknown reason.

deferSleep Initially set to 0. If a notification handler wants to defer sleep, then it should increment this value. When deferSleep is greater than 0, the system waits before going to sleep.

Compatibility Implemented only if Notification Feature Set is present.
sysNotifySyncFinishEvent
The sysNotifySyncFinishEvent is broadcast immediately after a HotSync operation has completed. Register for this notification if you need to perform post-processing after HotSync operations.

sysNotifySyncFinishEvent Specific Data
None.

Compatibility
Implemented only if Notification Feature Set is present.

sysNotifySyncStartEvent
The sysNotifySyncStartEvent is broadcast immediately before a HotSync operation is begun. Register for this notification if you need to perform preprocessing before a HotSync operation.

sysNotifySyncStartEvent Specific Data
None.

Compatibility
Implemented only if Notification Feature Set is present.

sysNotifyTimeChangeEvent
The sysNotifyTimeChangeEvent notification is broadcast just after the system time has been changed using TimSetSeconds. Register for this notification if you need to know when the time has changed.

sysNotifyTimeChangeEvent Specific Data
None.

Compatibility
Implemented only if Notification Feature Set is present.

sysNotifyVirtualCharHandlingEvent
The sysNotifyVirtualCharHandlingEvent is broadcast to enable custom handling of virtual characters.
Notifications
Notification Reference

sysNotifyVirtualCharHandlingEvent Specific Data

notifyDetailsP points to a SysNotifyVirtualCharHandlingType structure.

Prototype
typedef struct SysNotifyVirtualCharHandlingTag{
    UInt32 version;
    struct _KeyDownEventType keyDown;
} SysNotifyVirtualCharHandlingType;

Fields
version       The current version of this structure. The current version is 0.
keyDown       The virtual character. See the description of the keyDownEvent in the Event Reference section of the Palm OS Programmer’s API Reference for a complete description of this structure and its contents.

Compatibility
This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don’t broadcast this notification. Palm OS 5 does broadcast it. Later versions may or may not broadcast this notification.

sysNotifyVolumeMountedEvent

The sysNotifyVolumeMountedEvent is broadcast when a Virtual File System Manager volume is mounted. When a volume is mounted, the VFS Manager activates the start.prc application on the newly mounted volume and switches applications to the Launcher or to the start.prc application on that volume if it has a user interface.

Register for this notification if you need to know when a volume is mounted or if you want to prevent the default behavior of the VFS Manager.

To prevent the VFS Manager from activating the start.prc application, set the vfsHandledStartPrc bit in the handled field. To prevent the VFS Manager from switching applications, set the vfsHandledUIAppSwitch bit.
**sysNotifyVolumeMountedEvent Specific Data**

`notifyDetailsP` points to a `VFSSlotMountParamType` or `VFSPoseMountParamType` structure.

**Compatibility**

Implemented only if [4.0 New Feature Set](#) is present.

**sysNotifyVolumeUnmountedEvent**

The `sysNotifyVolumeUnmountedEvent` is broadcast when a Virtual File System Manager volume is unmounted. Register for this notification if you need to know when a volume is unmounted.

**sysNotifyVolumeUnmountedEvent Specific Data**

`notifyDetailsP` points to a `UInt16` containing the volume reference number.

**Compatibility**

Implemented only if [4.0 New Feature Set](#) is present.
Attention Manager

This chapter provides reference material for the Attention Manager, and is divided into the following sections:

- Attention Manager Data Structures
- Attention Manager Constants
- Attention Manager Functions
- Application-Defined Functions

The Attention Manager API is declared in the header file AttentionMgr.h.

For more information about the attention manager, see the section “Getting the User’s Attention” in the Palm OS Programmer’s Companion, vol. I.

**IMPORTANT:** The Attention Manager was introduced in Palm OS® 4.0 and is not available in earlier versions of the operating system.

### Attention Manager Data Structures

**AttnCommandType**

The AttnCommandType typedef specifies the set of possible commands that can be sent to the application requesting the user’s attention, either as a parameter to the AttnCallbackProc callback function or as one of the arguments that accompanies a sysAppLaunchCmdAttention launch code.

    typedef UInt16 AttnCommandType;

The following table lists the values that AttnCommandType can assume.
<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kAttnCommandDrawDetail</td>
<td>Indicates that the application needs to draw the detailed contents of the attention dialog. The command arguments parameter points to a structure of type drawDetail.</td>
</tr>
<tr>
<td>kAttnCommandDrawList</td>
<td>Indicates that the application needs to draw the appropriate list item in the attention dialog. The command arguments parameter points to a structure of type drawList.</td>
</tr>
<tr>
<td>kAttnCommandPlaySound</td>
<td>Indicates that the application needs to play a sound. The command arguments parameter is NULL.</td>
</tr>
<tr>
<td>kAttnCommandCustomEffect</td>
<td>Indicates that the application needs to perform any application-specific special effects. This command is only sent for attention items that set the kAttnFlagsCustomEffectBit when they call AttnGetAttention, which most applications won’t do.</td>
</tr>
<tr>
<td>kAttnCommandGoThere</td>
<td>Tells the application to navigate to the item. The command arguments parameter is NULL. An application commonly calls SysAppLaunch upon receipt of this command to have itself launched.</td>
</tr>
<tr>
<td>kAttnCommandGotIt</td>
<td>Tells the application that the user is dismissing the item. The command arguments parameter points to a structure of type gotIt. The application may choose to clean up memory at this point.</td>
</tr>
</tbody>
</table>
The `AttnCommandArgsType` structure is a union of C structures. How you interpret the union’s contents depends on which command it accompanies. Not all commands are accompanied by an `AttnCommandArgsType` structure, as listed in the following table:

<table>
<thead>
<tr>
<th><code>AttnCommandType</code></th>
<th>Accompanied By</th>
</tr>
</thead>
<tbody>
<tr>
<td>kAttnCommandDrawDetail</td>
<td><code>drawDetail</code></td>
</tr>
<tr>
<td>kAttnCommandDrawList</td>
<td><code>drawList</code></td>
</tr>
<tr>
<td>kAttnCommandPlaySound</td>
<td>None</td>
</tr>
<tr>
<td>kAttnCommandCustomEffect</td>
<td>None</td>
</tr>
<tr>
<td>kAttnCommandGoThere</td>
<td>None</td>
</tr>
<tr>
<td>kAttnCommandGotIt</td>
<td><code>gotIt</code></td>
</tr>
<tr>
<td>kAttnCommandSnooze</td>
<td>None</td>
</tr>
<tr>
<td>kAttnCommandIterate</td>
<td><code>iterate</code></td>
</tr>
</tbody>
</table>
Attention Manager
Attention Manager Data Structures

The structures that make up the AttnCommandArgsType union are described in the following sections.

drawDetail

When kAttnCommandDrawDetail is passed to the application, either via the callback function or as a parameter accompanying the sysAppLaunchCmdAttention launch code, the application needs to draw the detailed contents of the attention dialog. The drawDetail structure accompanies the kAttnCommandDrawDetail command, and provides the information needed to draw the contents of that dialog.

```c
struct AttnCommandArgsDrawDetailTag {
    RectangleType bounds;
    Boolean firstTime;
    AttnFlagsType flags;
} drawDetail;
```
Field Descriptions

bounds
Contains the window-relative bounding box for the area to draw. The clipping region is also set to the dimensions of this box to prevent accidental drawing outside.

firstTime
Set to true if the user has not yet seen this item. The value of this field could be used to display attentions that the user hasn’t seen before in some special way. firstTime also indicates to your application whether or not the area in which your application is to draw has been erased. If firstTime is false, the area is not guaranteed to be blank; your application will need to erase it.

flags
The global user preferences for this attention attempt combined with the custom flags passed in by the developer. Only the lower 16 bits of this field have meaning; use kAttnFlagsSoundBit, kAttnFlagsLEDBit, kCustomFlagsVibrateBit, and kAttnFlagsCustomEffectBit (described under “AttnFlagsType” on page 111) to interpret the contents of this field.

drawList
When kAttnCommandDrawList is passed to the application, either via the callback function or as a parameter accompanying the sysAppLaunchCmdAttention launch code, the application needs to draw the appropriate list item in the attention dialog. The drawList structure accompanies the kAttnCommandDrawList command, and provides the information needed to draw the contents of that dialog.

```
struct AttnCommandArgsDrawListTag {
    RectangleType bounds;
    Boolean firstTime;
    AttnFlagsType flags;
```
Boolean selected;
} drawList;

Field Descriptions

bounds        Contains the window-relative bounding box for the area to draw. The clipping region is also set to the dimensions of this box to prevent accidental drawing outside.

firstTime     Set to true if the user has not yet seen this item. The value of this field could be used, for example, to trigger a custom sound the first time this attention item is presented to the user.

flags         The global user preferences for this attention attempt combined with the custom flags passed in by the developer. Only the lower 16 bits of this field have meaning; use kAttnFlagsSoundBit, kAttnFlagsLEDBit, kCustomFlagsVibrateBit, and kAttnFlagsCustomEffectBit (described under “AttnFlagsType” on page 111) to interpret the contents of this field.

selected      Set to true if the item has been selected. It is up to your code to draw the item appropriately (typically by changing the UI colors) based upon the value of this flag.

gotIt

When kAttnCommandGotIt is passed to the application, either via the callback function or as a parameter accompanying the sysAppLaunchCmdAttention launch code, it is accompanied by an gotIt structure. This structure indicates whether the kAttnCommandGotIt command was generated because the user dismissed the attention, or whether the system is simply informing your application that AttnForgetIt was called. Your application normally ignores the latter case if your application made the call to AttnForgetIt.
struct AttnCommandArgsGotItTag {
    Boolean dismissedByUser;
} gotIt;

Field Descriptions

dismissedByUser  true indicates that the user dismissed the attention. false indicates that the kAttnCommandGotIt command was generated by a call to AttnForgetIt.

iterate

When kAttnCommandIterate is passed to the application, either via the callback function or as a parameter accompanying the sysAppLaunchCmdAttention launch code, it is accompanied by an iterate structure. This structure contains any necessary data that the application may need in order to process the callback or launch code.

struct AttnCommandArgsIterateTag {
    UInt32 iterationData;
} iterate;

Field Descriptions

iterationData  Any necessary data that the application may need in order to process the callback or launch code. The value of this field is that which was originally passed to AttnIterate.

AttnFlagsType

Pass a value of this type to AttnGetAttention and AttnUpdate to specify what means the device should or should not use to get the user’s attention. A value of this type is also passed back to your code either as a parameter to the callback function or, if no callback was specified, as part of the structure passed with the sysAppLaunchCmdAttention launch code.
Attention Manager
Attention Manager Data Structures

typedef UInt32 AttnFlagsType;
Note that more bits may be defined if necessary to accommodate additional hardware.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kAttnFlagsSoundBit</td>
<td>0x0001</td>
<td>Plays a sound.</td>
</tr>
<tr>
<td>kAttnFlagsLEDBit</td>
<td>0x0002</td>
<td>Blinks an LED, if the device is so equipped.</td>
</tr>
<tr>
<td>kAttnFlagsVibrateBit</td>
<td>0x0004</td>
<td>Triggers vibration, if the device is so equipped.</td>
</tr>
<tr>
<td>kAttnFlagsCustomEffectBit</td>
<td>0x0008</td>
<td>Triggers an application-specific custom effect.</td>
</tr>
<tr>
<td>kAttnFlagsAllBits</td>
<td>0xFFFF</td>
<td>Uses all available means to get the user’s attention.</td>
</tr>
<tr>
<td>kAttnFlagsUseUserSettings</td>
<td>0x0000</td>
<td>System-wide preferences determine what means are used to get the user’s attention.</td>
</tr>
</tbody>
</table>

The following constant values can be used to override the user’s settings and force or prevent specific behaviors:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kAttnFlagsAlwaysSound</td>
<td>kAttnFlagsSoundBit</td>
<td>Play a sound, regardless of the user’s settings.</td>
</tr>
<tr>
<td>kAttnFlagsAlwaysLED</td>
<td>kAttnFlagsLEDBit</td>
<td>Blink an LED, if the device is so equipped, regardless of the user’s settings.</td>
</tr>
<tr>
<td>kAttnFlagsAlwaysVibrate</td>
<td>kAttnFlagsVibrateBit</td>
<td>Vibrate, if the device is so equipped, regardless of the user’s settings.</td>
</tr>
<tr>
<td>kAttnFlagsAlwaysCustomEffect</td>
<td>kAttnFlagsCustomEffectBit</td>
<td>Trigger an application-specific custom effect.</td>
</tr>
</tbody>
</table>
These constants can be used in combination. For example, to disable both sound and the LED, use
\[ k\text{AttnFlagsNoSound} \mid k\text{AttnFlagsNoLED} \].

If neither \( k\text{AttnFlagsAlwaysSound} \) nor \( k\text{AttnFlagsNoSound} \) is set for a given attention item, a sound plays if and only if the user’s preference is to play a sound and the user’s preference for alarm volume is non-zero.

### AttnLaunchCodeArgsType

If a callback function is not specified in a call to \( \text{AttnGetAttention} \) and the Attention Manager needs your code to draw the details of your attention in the attention dialog or perform another attention-specific function, it sends a
Attention Manager
Attention Manager Data Structures

sysAppLaunchCmdAttention launch code to your application. Along with the launch code, it passes a pointer to the following structure, which indicates both what your code is expected to do and identifies the attention that triggered the launch code:

```c
typedef struct {
    AttnCommandType command;
    UInt32 userData;
    AttnCommandArgsType *commandArgsP;
} AttnLaunchCodeArgsType;
```

Field Descriptions

**command**
Indicates what your code is being requested to do. The complete list of possible commands are described in the definition of AttnCommandType.

**userData**
Identifier that distinguishes the particular attention item from others made by this application. This identifier was specified when the attention item was created.

**commandArgsP**
Pointer to command-specific arguments. See the description of each command for a discussion of that command’s arguments.

When processing the launch code be aware that your application doesn’t have application globals available to it; it is important that anything necessary to draw or otherwise display be available through commandArgsP.

**AttnLevelType**
Attention attempts can either be insistent or subtle. Insistent attention attempts make a serious effort to get the user’s attention, by both displaying a dialog and possibly by triggering one or more special effects, such as blinking a light, vibrating, or playing a sound. Other alerts are of a less serious nature and shouldn’t disrupt the user. Consequently, subtle attention attempts typically make the attention indicator blink and may trigger one or more special effects, but don’t display the Attention Manager dialog. The user can then work until a suitable time, at which point they can tap
on the indicator to see what needs their attention. Subtle attention attempts might be used for telling the user that they have new e-mail, or perhaps that a holiday or birthday is coming up.

```c
typedef UINT16 AttnLevelType;
```

The following table lists the two defined values for `AttnLevelType`:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kAttnLevelInsistent</td>
<td>An insistent attention attempt. Make a serious effort to get the user’s attention by displaying a dialog and optionally triggering one or more special effects.</td>
</tr>
<tr>
<td>kAttnLevelSubtle</td>
<td>A subtle attention attempt. Notify the user using special effects, but don’t disrupt the user with the dialog if the device is in use.</td>
</tr>
</tbody>
</table>

Note that user preferences for the various special effects can’t be set separately for subtle and insistent attention attempts. If your application needs to vary the effects based upon the `AttnLevelType`, pass a suitable value for the `flags` parameter in your `AttnGetAttention` call.

**Attention Manager Constants**

In addition to the constant values defined specifically for use with `AttnCommandType`, `AttnFlagsType`, and `AttnLevelType`, the Attention Manager defines the following constant types:

- **Error Code Constants**
- **Attention Manager Drawing Constants**
- **Attention Manager Feature Constants**

**Error Code Constants**

The Attention Manager returns the following error code under the circumstances described below.
Attention Manager
Attention Manager Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attnErrMemory</td>
<td>Returned by AttnGetAttention when there is insufficient memory to perform the requested operation.</td>
</tr>
</tbody>
</table>

Attention Manager Drawing Constants
The following four constants define the on-screen boundaries of the attention indicator:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kAttnIndicatorLeft</td>
<td>0</td>
<td>The left-hand edge of the attention indicator.</td>
</tr>
<tr>
<td>kAttnIndicatorTop</td>
<td>0</td>
<td>The top-most edge of the attention indicator.</td>
</tr>
<tr>
<td>kAttnIndicatorWidth</td>
<td>16</td>
<td>The width of the attention indicator.</td>
</tr>
<tr>
<td>kAttnIndicatorHeight</td>
<td>16</td>
<td>The height of the attention indicator.</td>
</tr>
</tbody>
</table>

The following two constants are used when drawing the list view. Applications should use these constants to format the display of information in the Attention Manager’s list view. Draw the application’s small icon centered within the first kAttnListMaxIconWidth pixels of the drawing bounds. Then draw two lines of text describing the attention, left-justified, starting at kAttnListTextOffset from the left edge of the drawing bounds.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kAttnListMaxIconWidth</td>
<td>15</td>
<td>Maximum width of the application’s icon. If the icon is narrower than this, it should be drawn centered within this width.</td>
</tr>
<tr>
<td>kAttnListTextOffset</td>
<td>17</td>
<td>Offset, from the left-hand edge of the drawing bounds, of the textual description of the attention.</td>
</tr>
</tbody>
</table>
Attention Manager Feature Constants

The Attention Manager defines a read-only feature ('attn', 0) that indicates the current user settings and capabilities of the hardware. The upper 16 bits of the feature indicate whether or not the hardware has the capability to perform that sort of alert. The lower 16 bits indicate whether the user has currently enabled that sort of alert.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kAttnFtrCreator</td>
<td>'attn'</td>
<td>Attention Manager feature creator.</td>
</tr>
<tr>
<td>kAttnFtrCapabilities</td>
<td>0</td>
<td>Attention Manager feature number.</td>
</tr>
</tbody>
</table>

When working with the value obtained with FtrGet, use the following two constants to separate those bits that contain the user settings from those bits that identify the device’s capabilities:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kAttnFlagsUserSettingsMask</td>
<td>kAttnFlagsAllBits</td>
<td>Mask to isolate those bits that contain the user settings.</td>
</tr>
<tr>
<td>kAttnFlagsCapabilitiesMask</td>
<td>kAttnFlagsAllBits &lt;&lt; 16</td>
<td>Mask to isolate those bits that contain the device capabilities.</td>
</tr>
</tbody>
</table>

These constants can be used to interpret the device capabilities (kAttnFlagsHas...) and the user settings (kAttnFlagsUserWants...):
## Attention Manager
### Attention Manager Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kAttnFlagsHasLED</td>
<td>kAttnFlagsLEDBit &lt;&lt; 16</td>
<td>The device has an LED that can be illuminated to indicate an alert.</td>
</tr>
<tr>
<td>kAttnFlagsHasSound</td>
<td>kAttnFlagsSoundBit &lt;&lt; 16</td>
<td>The device is capable of playing a sound to indicate an alert.</td>
</tr>
<tr>
<td>kAttnFlagsHasVibrate</td>
<td>kAttnFlagsVibrateBit &lt;&lt; 16</td>
<td>The device is capable of vibrating to indicate an alert.</td>
</tr>
<tr>
<td>kAttnFlagsHasCustomEffect</td>
<td>kAttnFlagsCustomEffectBit &lt;&lt; 16</td>
<td>Not used.</td>
</tr>
<tr>
<td>kAttnFlagsUserWantsLED</td>
<td>kAttnFlagsLEDBit</td>
<td>The user wants the LED illuminated to signal an alert.</td>
</tr>
<tr>
<td>kAttnFlagsUserWantsSound</td>
<td>kAttnFlagsSoundBit</td>
<td>The user wants a sound played to signal an alert.</td>
</tr>
<tr>
<td>kAttnFlagsUserWantsVibrate</td>
<td>kAttnFlagsVibrateBit</td>
<td>The user wants the device to vibrate to signal an alert.</td>
</tr>
<tr>
<td>kAttnFlagsUserWantsCustomEffect</td>
<td>kAttnFlagsCustomEffectBit</td>
<td>Not used.</td>
</tr>
</tbody>
</table>
Attention Manager Functions

**AttnDoSpecialEffects**

**Purpose**
Triggers an Attention Manager special effect set.

**Declared In**
AttentionMgr.h

**Prototype**
Err AttnDoSpecialEffects (AttnFlagsType flags)

**Parameters**
-> flags
Specifications the behavior to be exhibited by this special effects request. See AttnFlagsType for the various bits that make up this flag. Note that the behavior is undefined if you set incompatible flags. Supply kAttnFlagsUseUserSettings to have this attention request follow the user’s pre-set preferences.

**Result**
Returns errNone if no problems were encountered. Returns attnErrMemory if there wasn’t enough memory to accommodate the attention request.

**Comments**
This routine is provided as a convenience for applications that need to trigger special effects. It does the equivalent of one “nag” of an Attention Manager special effect set.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.
AttnForgetIt

Purpose Provides a way for applications to tell the Attention Manager to forget about an attention item.

Declared In AttentionMgr.h

Prototype Boolean AttnForgetIt (UInt16 cardNo, LocalID dbID, UInt32 userData)

Parameters
-> cardNo Card number on which the application making the request resides.
-> dbID Database ID of the application making the request.
-> userData Identifier that distinguishes the attention attempt from others made by the same application. This identifier can be an integer, a pointer, or any other 32-bit value.

Result Returns true if the item was removed, false if a matching item was not found.

Comments You typically call this function after your application has handled a “Go There” event and the user has read about the item. For example, if there is a subtle attention pending that says “you have three e-mail messages waiting” and you go to the e-mail application on your own and read your e-mail, the subtle notification must disappear. AttnForgetIt allows the application to do this.

Note that this call can be made when the Attention Manager dialog is on-screen (though presumably that is rare, since the application is probably not doing much at this point). If this call removes a list item, then the Attention Manager may call back into other items to redraw the list.

If this call removes the last item when any indicator is present, the indicator disappears. If this call removes the last unread item, but read items remain, the indicator switches from blinking to steady state.
Compatibility  Implemented only if 4.0 New Feature Set is present.

**AttnGetAttention**

**Purpose**  Requests the user’s attention.

**Declared In**  AttentionMgr.h

**Prototype**  

```
Err AttnGetAttention (UInt16 cardNo, 
LocalID dbID, UInt32 userData, 
AttnCallbackProc *callbackFnP, 
AttnLevelType level, AttnFlagsType flags, 
UInt16 nagRateInSeconds, UInt16 nagRepeatLimit)
```

**Parameters**

- **cardNo**  Card number on which the application making the request resides.
- **dbID**  Database ID of the application making the request.
- **userData**  Application-specific data that is later passed back to your code through the callback function. If no callback function is specified in the callbackFnP parameter, this data is included in what is passed along with a sysAppLaunchCmdAttention launch code. userData can be an integer, a pointer, or any other 32-bit value as needed by your application. Most applications pass the unique ID or other key for the record which caused the attention request. userData is also used to distinguish a given attention attempt from others made by the same application.
Attention Manager
Attention Manager Functions

-> callbackFnP
   Pointer to the function registered by the application to be called by the Attention Manager when the attention is displayed or removed. See AttnCallbackProc, below, for the callback function’s parameters. Supply NULL to instead have a sysAppLaunchCmdAttention launch code sent to the application that made the attention request whenever the attention is displayed or removed.

-> level
   Indicates the annoyance level. Pass one of the values defined for AttnLevelType.

-> flags
   Behavior override, if necessary, for this attention request. This override allows, for instance, silent alarms or noisy alarms. See AttnFlagsType for the various bits that make up this flag. Note that the behavior is undefined if you set incompatible flags. Supply kAttnFlagsUseUserSettings to have this attention request follow the user’s pre-set preferences.

-> nagRateInSeconds
   How long to wait before nagging.

-> nagRepeatLimit
   How many times to nag, excluding the first attempt.

Result
Returns errNone if no problems were encountered. Returns attnErrMemory if there wasn’t enough memory to accommodate the attention request.

Comments
The combination of cardNo, dbID and userData uniquely identify an attention-getting attempt. If another call is made to AttnGetAttention with identical values for these arguments, an error is reported. To update or delete an existing attention item, pass these same values to AttnUpdate or AttnForgetIt, respectively.
In response to AttnGetAttention, the behavior of the operating system or application depends on whether there already are other demands and on the annoyance level passed in the AttnGetAttention call.

- No other demands, insistent attention request:
  The Attention Manager puts up a dialog that details the current attempt to get the user’s attention.

- Other demands exist, insistent attention request:
  The Attention Manager adds a summary of the current attempt to get the user’s attention to a list of things that need attention. If the dialog is currently in detail form—which is the case if just one other demand exists—the view is refreshed, changing from detail to list form. In this case, the pen and key event queues are also flushed so that any user events that are happening while the display is changing are explicitly ignored. Two exceptions to this behavior exist: if all existing attentions are subtle, or if all existing insistent attentions were snoozed, the new insistent attention brings up the dialog in detail mode, rather than list mode.

- Subtle attention request:
  The Attention Manager starts the attention indicator blinking, and adds the item to its list for later display, unless the dialog is currently being displayed in list mode. In this event, the new subtle attention item simply appears in the list; the indicator does not blink to announce it.

**Compatibility**

Implemented only if [4.0 New Feature Set](#) is present.

**See Also**

AttnUpdate
AttnGetCounts

**Purpose**

Returns the number of attention items that are currently pending.

**Declared In**

AttentionMgr.h

**Prototype**

```c
UInt16 AttnGetCounts (UInt16 cardNo, LocalID dbID, UInt16 *insistentCountP, UInt16 *subtleCountP)
```

**Parameters**

- **cardNo**
  
  If this value is zero, counts pending attention items from applications on all cards. Otherwise, counts only pending attention items from applications on the specified card.

- **dbID**
  
  If this value is zero, counts pending attention items from all applications. Otherwise, counts only pending attention items from applications with the specified database ID.

- **insistentCountP**
  
  Pointer to a 16-bit unsigned value that is filled in with the number of insistent items pending. Pass NULL for this parameter if you don’t need to know the number of insistent items that are pending.

- **subtleCountP**
  
  Pointer to a 16-bit unsigned value that is filled in with the number of subtle items pending. Pass NULL for this parameter if you don’t need to know the number of subtle items that are pending.

**Result**

Returns the total number of items, both insistent and subtle, that are currently pending.

**Comments**

Call this function if you need to exhibit different behavior if attention items are already pending.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.
**AttnIndicatorEnable**

**Purpose** Enables and disables the on-screen attention indicator.

**Declared In** AttentionMgr.h

**Prototype**

```c
void AttnIndicatorEnable (Boolean enableIt)
```

**Parameters**

`enableIt` true to enable the attention indicator, false to disable it.

**Result** Returns nothing.

**Comments**

This function is used by applications to enable or disable the on-screen attention indicator. The indicator only blinks when all of the following are true:

- The indicator is enabled.
- The indicator is being asked to blink by the attention manager.
- The operating system isn’t using the display in such a way as to prevent the attention indicator from showing, such as when the menu bar is being displayed or when a modal dialog is on top of the form.

The attention indicator is enabled by default. If your application controls the upper portion of the screen and needs to prevent the attention indicator from being displayed, call `AttnIndicatorEnable` and pass it a value of false.

**Compatibility** Implemented only if 4.0 New Feature Set is present.

**See Also** `AttnIndicatorEnabled`
AttnIndicatorEnabled

**Purpose**
Returns whether the on-screen attention indicator is currently enabled.

**Declared In**
AttentionMgr.h

**Prototype**
Boolean AttnIndicatorEnabled (void)

**Parameters**
None.

**Result**
Returns true if the on-screen attention indicator is currently enabled, false otherwise.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
AttnIndicatorEnable

AttnIterate

**Purpose**
Instructs the Attention Manager to check each attention item currently pending and, for those that match the specified card number and database ID, invoke the item’s callback routine. If a callback routine was not specified in the request, the sysAppLaunchCmdAttention launch code is sent to the application that made the attention request.

**Declared In**
AttentionMgr.h

**Prototype**
void AttnIterate (UInt16 cardNo, LocalID dbID, UInt32 iterationData)

**Parameters**
-> cardNo
Card number on which the application that made the request resides.

-> dbID
Database ID of the application that made the request.
-> iterationData
Any necessary data that the application may
need in order to process the callback or launch
code. See the description of the
AttnCallbackProc function for more
information on this parameter.

Result
Returns nothing.

Comments
This function iterates through all of the attention requests made by
this application and uses the callback or launch code for each to
inform the application about the attention request. When an
application receives a sysAppLaunchCmdSyncNotify launch
code, signifying that a HotSync occurred that affected that
application's databases, the application typically calls
AttnIterate so it can remove attention requests for records that
may have been removed during the HotSync. Applications can also
call AttnGetAttention after a HotSync, if necessary.

Note that you can call AttnForgetIt inside the iteration since it
only marks the record for deletion and thus doesn’t confuse the
iteration.

Compatibility
Implemented only if 4.0 New Feature Set is present.

AttnListOpen

Purpose
Displays the attention dialog in list mode and, after the user has
dismissed it, acts accordingly based on how it was dismissed.

Declared In
AttentionMgr.h

Prototype
void AttnListOpen (void)

Parameters
None.

Result
Returns nothing.
Attention Manager
Attention Manager Functions

Comments
This function allows applications that do not provide the blinking attention indicator to open the list, if necessary.

Compatibility
Implemented only if 4.0 New Feature Set is present.

AttnUpdate

Purpose
Updates one or more aspects of a specified attention item.

Declared In
AttentionMgr.h

Prototype
Boolean AttnUpdate (UInt16 cardNo, LocalID dbID, UInt32 userData, AttnCallbackProc *callbackFnP, AttnFlagsType *flagsP, UInt16 *nagRateInSecondsP, UInt16 *nagRepeatLimitP)

Parameters
- `-> cardNo`  Card number on which the application that made the request resides.
- `-> dbID`  Database ID of the application that made the request.
- `-> userData`  Application-specific data that is passed back to your code through the callback function. If no callback function is specified in the callbackFnP parameter, this data is included in what is passed along with a sysAppLaunchCmdAttention launch code. userData can be an integer, a pointer, or any other 32-bit value. Most applications pass the unique ID or other key for the record which caused the attention request. The value of the userData parameter is also used to distinguish a given attention attempt from others made by the same application.

128  Palm OS Programmer’s API Reference
-> callbackFnP

Registers a new function to be called by the Attention Manager when the attention is displayed or removed. The function to which this parameter points should conform to `AttnCallbackProc`. Supply `NULL` to instead have a `sysAppLaunchCmdAttention` launch code sent to the application that made the attention request whenever the attention is displayed or removed.

**IMPORTANT:** Because `NULL` indicates that a launch code should be sent whenever the callback would otherwise be invoked, it isn’t used in this instance to leave the original setting for the `callbackFnP` parameter intact. The value supplied for this parameter *always* overwrites the value supplied in the original attention request.

-> flagsP

Pointer to a set of flags that can be used to override user-specified attention behavior; for instance, to force silent or noisy alarms. See `AttnFlagsType` for the various bits that make up this flag, and note that the behavior is undefined if you set incompatible flags. Pass `NULL` to leave the current flag settings unchanged.

-> nagRateInSecondsP

Pointer to the length of time to wait before nagging. Pass `NULL` to leave the “nag rate” unchanged.

-> nagRepeatLimitP

Pointer to the maximum number of times the user should be nagged. Pass `NULL` to leave the nag repeat limit unchanged.

**Result** Returns `true` if the update was successful, `false` if no matching attention item was found.
Comments

This call may result in the callback function being called to re-display the item. If no callback function is specified, the `sysAppLaunchCmdAttention` launch code is instead sent to your application. It may also result in callbacks to other pending attention requests.

You call `AttnUpdate` to tell the Attention Manager to update, forcing it to call into all of its clients to redraw. This provides a way for an application to update the text of an attention item without tearing down and re-opening the Attention Manager dialog. For example, `AttnUpdate` could be used to update an existing email notification to say “You have three new email messages” when additional messages are received.

Although `AttnUpdate` may cause a given attention item to redraw, it does not rerun the special effects (if any) that occurred when that attention item was added. If you want to trigger Attention Manager effects for a particular item, call `AttnForgetIt` followed by `AttnGetAttention`.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`AttnGetAttention`

Application-Defined Functions

`AttnCallbackProc`

Purpose

Provides a function prototype to be used by callback functions supplied to `AttnGetAttention` and `AttnUpdate`. The supplied
function is invoked by the Attention Manager whenever the attention is displayed or removed.

Declared In  AttentionMgr.h

Prototype  

typedef Err AttnCallbackProc
(AttnCommandType command, UInt32 userData,
AttnCommandArgsType *commandArgsP)

Parameters

- **-> command**  Indicates what the callback function is being requested to do. The complete list of possible commands are described in the definition of AttnCommandType.

- **-> userData**  Identifier that distinguishes the particular attention item from others made by this application. This identifier was specified when the attention item was created.

- **-> commandArgsP**  Pointer to command-specific arguments. See the description of each command for a discussion of that command’s arguments.

Result  The callback function should return errNone if it correctly handled the command, or an appropriate error code otherwise. If the callback function returns an error code other than errNone, the attention is removed from the list of active attention items.

Comments  For a given attention item, the Attention Manager calls back to the code resource that created that item whenever the Attention Manager needs the resource to draw the attention dialog contents or whenever it needs to inform the code resource of activity relating to the attention item. The Attention Manager calls back using one of two mechanisms:

- If a callback routine has been specified for a given attention item, the Attention Manager invokes the specified routine. This callback routine doesn’t have application globals available to it, so it is important that anything necessary to draw or otherwise display be available through
Attention Manager
Application-Defined Functions

commandArgsP. A callback routine is typically used by libraries and system extensions.

- If a callback routine has not been specified for a given attention item, the Attention Manager instead sends a `sysAppLaunchCmdAttention` launch code to the application that registered the attention item. Accompanying that launch code is an `AttnLaunchCodeArgsType` structure containing the three parameters documented above. Applications typically use the launch-code mechanism due to the restrictions that are placed on callback routines.

**IMPORTANT:** It is your responsibility to ensure that the callback procedure is still in the same place when it gets called, dealing with the possibility that the code resource might be unlocked and moved in memory, and with the possibility that the database containing the code resource might be deleted. For the most part, these problems don’t exist when using launch codes.

**Compatibility**  Invoked only if 4.0 New Feature Set is present.
Categories

This chapter describes the category API as declared in the header file Category.h. It discusses the following topics:

- **Category Data Structures**
- **Category Constants**
- **Category Functions**

For more information on categories, see the section “Categories” on page 116 in the Palm OS Programmer’s Companion, vol. I.

Category Data Structures

**AppInfoPtr**

The AppInfoPtr defines a pointer to an AppInfoType structure.

```c
typedef AppInfoType *AppInfoPtr;
```

**AppInfoType**

The AppInfoType structure shown below maps category names to category indexes and unique IDs. To use the category API described in this chapter, a database’s application info block must either be an AppInfoType structure, or it must have an AppInfoType structure as its first field.

```c
typedef struct {
    Uint16   renamedCategories;
    Char    categoryLabels [dmRecNumCategories]
            [dmCategoryLength];
    Uint8    categoryUniqIDs[dmRecNumCategories];
    Uint8    lastUniqID;
    Uint8    padding;
} AppInfoType;
```
Allocate the application info block in the storage heap and use the `DmSetDatabaseInfo` function to set the database’s application info ID to the local ID of this structure. Then, use the `CategoryInitialize` function to initialize it with a localized list of strings containing the category names.

### Field Descriptions

- **renamedCategories**
  Used by `CategorySetName` as a bit field indicating which categories have been renamed. Usually cleared by a conduit.

- **categoryLabels**
  An array of strings containing the category names. The maximum size of the array is `dmRecNumCategories`, and the maximum length of each string in the array is `dmCategoryLength`. Both of these constants are defined in `DataMgr.h`.

- **categoryUniqIDs**
  Category IDs used for synchronization with the desktop database. Unique IDs generated by the device are between 0 and 127. Unique IDs generated by the desktop computer are between 128 and 255.

- **lastUniqID**
  Used for sorting and assigning unique IDs.

### Category Constants

The following category constants are defined:
NOTE: These constants look like system resource IDs, but they are not. To use a non-default string for the “Edit Categories” item you pass a resource ID of a string containing your title. If you want to use the default or hide the item, you pass one of these constants. They are within the system resource ID range (that is, they are greater than 10000) so that they don’t conflict with any other possible value for that parameter.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>categoryHideEditCategory</td>
<td>10000</td>
<td>Used to suppress the “Edit Categories” item.</td>
</tr>
<tr>
<td>categoryDefaultEditCategoryString</td>
<td>10001</td>
<td>Used to show the default “Edit Categories” item.</td>
</tr>
</tbody>
</table>

Compatibility: Both categoryHideEditCategory and categoryDefaultEditCategoryString are defined only if the 3.5 New Feature Set is present.
Category Functions

CategoryCreateList

**Purpose**
Populate a popup list with a database’s categories.

**Declared In**
Category.h

**Prototype**
```c
void CategoryCreateList (DmOpenRef db,
                        ListType *listP, UInt16 currentCategory,
                        Boolean showAll, Boolean showUneditables,
                        UInt8 numUneditableCategories,
                        UInt32 editingStrID, Boolean resizeList)
```

**Parameters**
- `-> db` Open database containing the category information you want to read.
- `<- listP` Pointer to the `ListType` structure that should display the categories.
- `-> currentCategory` Index of the category to select. The index is the index into the `categoryLabels` array. The default is to have the “Unfiled” category selected.
- `-> showAll` true to include an “All” list item.
- `-> showUneditables` true to show uneditable categories.
- `-> numUneditableCategories` The number of categories that the user cannot edit. You should store uneditable categories at the beginning of the `categoryLabels` array. For example, it’s common to have an “Unfiled” category at position zero that is not editable. This function displays the uneditable categories at the end of the popup list.
-> editingStrID  The resource ID of a tSTR resource to use as the Edit Categories list item. To use the default string (“Edit Categories”) pass the constant categoryDefaultEditCategoryString.

If you don’t want users to edit categories, pass the categoryHideEditCategory constant.

-> resizeList  true to resize the list to the number of categories. Set to true for popups, false otherwise.

Result  Returns nothing.

Comments  The “All” item is first in the list (if the showAll parameter is true), followed by the editable categories in the database and then the categories that cannot be edited. The option to edit categories is last in the list and can be suppressed if desired.

You rarely call this function directly. Instead, most applications use CategorySelect, which calls this function and fully manages the user’s selection of a category in the popup list. Use CategoryCreateList only if you want more control over the category popup list.

This function obtains the db parameter’s appInfoID, reads the AppInfoType structure at that location, and uses the information in it to initialize the listP’s items array with the names of the database’s categories. You must have already allocated the structure pointed to by listP. CategoryCreateList does not display the list; use LstPopupList or LstDrawList to do so.

You must balance a call to CategoryCreateList with a call to CategoryFreeList. The CategoryCreateList function locks the resources for the category names. It also allocates the listP items array. CategoryFreeList unlocks all resources locked by CategoryCreateList and frees all memory allocated by CategoryCreateList.

Compatibility  Implemented only if 2.0 New Feature Set is present.

The constants categoryDefaultEditCategoryString and categoryHideEditCategory are defined only if 3.5 New...
**Feature Set** is present. In earlier versions, you can suppress the Edit Categories string by passing 0 for the editingStrID parameter, or include the item by passing categoryEditStrID.

**See Also**  
CategoryCreateListV10

**CategoryCreateListV10**

**Purpose**  
Read a database’s categories and set the category list.

**Declared In**  
Category.h

**Prototype**  
```c
void CategoryCreateListV10 (DmOpenRef db, ListType *lst, UInt16 currentCategory, Boolean showAll)
```

**Parameters**

- `db`  
  Open database containing the category information you want to read.

- `lst`  
  Pointer to the ListType that should display the categories.

- `currentCategory`  
  Index of the category to select. The index is the index into the categoryLabels array. The default is to have the “Unfiled” category selected.

- `showAll`  
  true to include an “All” list item.

**Result**  
Returns nothing.

**Compatibility**  
This function corresponds to the Palm OS® 1.0 version of CategoryCreateList. It is obsolete.
CategoryEdit

Purpose  Event handler for the Edit Categories dialog.

Declared In  Category.h

Prototype  Boolean CategoryEdit (DmOpenRef db,
UInt16 *category, UInt32 titleStrID,
UInt8 numUneditableCategories)

Parameters

-> db  Open database containing the categories to be edited.

<- category  Upon return, the index of the last category selected before the dialog was closed.

-> titleStrID  The resource ID of a tSTR resource to use as the dialog’s title. To use the default string (“Edit Categories”), pass the constant categoryDefaultEditCategoryString.

-> numUneditableCategories  The number of categories that the user cannot edit. You should store uneditable categories at the beginning of the categoryLabels array. For example, it’s common to have an “Unfiled” category at position zero that is not editable.

Result  Returns true if any of the following conditions are true:

- The current category is renamed.
- The current category is deleted.
- The current category is merged with another category.

Comments  You rarely call this function directly. The CategorySelect function calls it when the user chooses the Edit Category list item.

This function both displays the Edit Categories dialog and handles the result of the user actions. It updates the AppInfoType structure’s list of categories and reassigns database records to new categories as needed. If a user deletes a category, CategoryEdit
moves all of the records belonging to that category to the Unfiled category. If a category is renamed to be the same as an existing category, this function moves all of the old category’s records to the new category.

**Compatibility**  Implemented only if [3.0 New Feature Set](#) is present.

**See Also**  [CategoryEditV20](#), [CategoryEditV10](#), [DmMoveCategory](#)

**CategoryEditV20**

**Purpose**  Event handler for the Edit Categories dialog.

**Declared In**  Category.h

**Prototype**  

```c
Boolean CategoryEditV20 (DmOpenRef db, UInt16 *category, UInt32 titleStrID)
```

**Parameters**

- `-> db`  Database containing the categories to be edited.
- `<- category`  Upon return, the last category selected before the dialog was closed.
- `-> titleStrID`  The resource ID of a tSTR resource to use as the dialog’s title.

**Result**  Returns true if any of the following conditions are true:

- The current category is renamed.
- The current category is deleted.
- The current category is merged with another category.

**Compatibility**  This function corresponds to the Palm OS 2.0 version of CategoryEdit. Implemented only if [2.0 New Feature Set](#) is present. This function is obsolete.

**See Also**  [CategoryEdit](#), [CategoryEditV10](#)
CategoryEditV10

Purpose Event handler for the Edit Categories dialog.

Declared In Category.h

Prototype Boolean CategoryEditV10 (DmOpenRef db, UInt16 *category)

Parameters -> db Open database containing the categories to be edited.
<- category Upon return, the last category selected before the dialog was closed.

Result Returns true if any of the following conditions are true:
• The current category is renamed.
• The current category is deleted.
• The current category is merged with another category.

Compatibility This function corresponds to the Palm OS 1.0 version of CategoryEdit. It is obsolete.

See Also CategoryEdit, CategoryEditV20

CategoryFind

Purpose Return the index of a category given its name.

Declared In Category.h

Prototype UInt16 CategoryFind (DmOpenRef db, const Char *name)

Parameters -> db Open database to search.
Categories
Category Functions

-> name Category name. Pass the empty string to find the first unused category.

Result Returns the index of the category’s entry in the categoryLabels array (see AppInfoType). Returns dmAllCategories if the category does not exist.

CategoryFreeList

Purpose Unlock or free memory locked or allocated by CategoryCreateList.

Declared In Category.h

Prototype void CategoryFreeList (DmOpenRef db, ListType *listP, Boolean showAll, UInt32 editingStrID)

Parameters
- db Open database containing the categories.
- listP Pointer to the category list. (See ListType.)
- showAll true if the list was created with an “All” category.
- editingStrID The resource ID that you passed as the editingStrID parameter to CategoryCreateList. This function unlocks the resource.

Result Returns nothing.

Comments You only need to call this function if you explicitly call CategoryCreateList. Typical applications call CategorySelect, which handles both the creation and deletion of the list.

This function frees the items in the popup list listP’s items array and it unlocks other resources that CategoryCreateList may have locked.
This function does not remove the categories from the passed database, and it does not free the ListType structure pointed to by listP. (Typically, a list is freed when its form is freed.)

**Compatibility**
Implemented only if [2.0 New Feature Set](#) is present.

**See Also**
CategoryFreeListV10

### CategoryFreeListV10

**Purpose**
Unlock or free memory locked or allocated by CategoryCreateListV10.

**Declared In**
Category.h

**Prototype**
```c
void CategoryFreeListV10 (DmOpenRef db, ListType *lst)
```

**Parameters**
- `-> db` Open database containing the categories.
- `-> listP` Pointer to the category list. (See ListType.)

**Result**
Returns nothing.

**Compatibility**
This function corresponds to the Palm OS 1.0 version of CategoryFreeList. It is obsolete.

**See Also**
CategoryFreeList
CategoryGetByName

**Purpose**
Return the name of the specified category.

**Declared In**
Category.h

**Prototype**
```c
void CategoryGetByName (DmOpenRef db, UInt16 index, Char *name)
```

**Parameters**
- `-> db` Database that contains the categories.
- `-> index` Category index. This is the index into the `categoryLabels` array in the `AppInfoType` structure. You can retrieve this index from a database record’s attribute word.
- `<- name` Buffer to hold category name. Buffer should be `dmCategoryLength` in size.

**Result**
Stores the category name in the `name` buffer passed.
May display a fatal error message if the index is out of range.

**Comments**
You can use this function to find out the name of a given database record’s category. Use the `DmRecordInfo` call to obtain the category index from the given record. For example:

```c
DmOpenRef myDB;
UInt16 record, attr, category;
Char *name;

DmRecordInfo(myDB, record, &attr, NULL, NULL);
category = attr & dmRecAttrCategoryMask;
CategoryGetByName(myDB, category, name);
```

The category’s name is copied into the variable you pass for the `name` parameter.

**See Also**
CategorySetName
CategoryGetNext

**Purpose**
Return the index of the next category after a given category.

**Declared In**
Category.h

**Prototype**
 UInt16 CategoryGetNext (DmOpenRef db, UInt16 index)

**Parameters**
- `db` Open database containing the categories.
- `index` Category index.

**Result**
Category index of next category.

**Comments**
The intended use of this function is to allow your users to cycle through categories. For example, the built-in applications cycle through categories when the user presses the corresponding hard-key button. (See the ListViewNextCategory function in the Address Book sample application for an example.) Note that categories are not displayed in the same order as they are stored. Do not use this function for searching for a particular category or iterating through a category list.

**Compatibility**
In Palm OS 1.0, the system chose Unfiled as one category.
In Palm OS 2.0 and later, the system skips both Unfiled and categories with empty records.
CategoryInitialize

**Purpose**
Initialize the category names, IDs, and flags.

**Declared In**
Category.h

**Prototype**
```c
void CategoryInitialize (AppInfoPtr appInfoP,
UInt16 localizedAppInfoStrID)
```

**Parameters**
- `->appInfoP` Pointer to the locked application info block. See `AppInfoType`.
- `->localizedAppInfoStrID` Resource ID of the localized category names. This must be a resource of the type `appInfoStringsRsc('tAIS')`.

**Result**
Returns nothing.

**Comments**
Call this function at database creation time to initialize the database’s categories from a list of localized strings.

CategoryInitialize initializes the `AppInfoType` structure that is associated with your database. It does not create the structure. To create the structure, you must allocate it in the storage heap (using `DmNewHandle`) and associate it with your database using `DmSetDatabaseInfo`.

**Compatibility**
Implemented only if 2.0 New Feature Set is present.
CategorySelect

Purpose Process the selection and editing of categories.

Declared In Category.h

Prototype

Boolean CategorySelect (DmOpenRef db, const FormType *frm, UInt16 ctlID, UInt16 lstID, Boolean title, UInt16 *categoryP, Char *categoryName, UInt8 numUneditableCategories, UInt32 editingStrID)

Parameters

- > db Open database containing the categories.
- > frm Form that contains the category popup list.
- > ctlID ID of the popup trigger.
- > lstID ID of the popup list.
- > title true to have an “All” list item. (In general, if the trigger is in the form’s title bar, it should have an “All” item. If the trigger is elsewhere in the form, it should not.)

<-> categoryP Index of the selected category. The index is the index into the categoryLabels array.

<-> categoryName Name of the selected category.

- > numUneditableCategories The number of categories that the user cannot edit. You should store uneditable categories at the beginning of the categoryLabels array. For example, it’s common to have an “Unfiled” category at position zero that is not editable. This function displays the uneditable categories at the end of the popup list.

- > editingStrID The resource ID of a tSTR resource to use as the Edit Categories list item. To use the default string (“Edit Categories”), pass the constant categoryDefaultEditCategoryString.
Categories
Category Functions

If you don’t want users to edit categories, pass the categoryHideEditCategory constant.

**Result**

Returns true if any of the following conditions are true:

- The current category is renamed.
- The current category is deleted.
- The current category is merged with another category.

**Comments**

Call this function when the user taps the category popup trigger. This function handles all aspects of displaying the popup list and managing the user selection—It creates the popup list using CategoryCreateList, displays the popup list, calls CategoryEdit if the user selects the Edit Categories item, uses CategorySetTriggerLabel to set the trigger label to the item the user selected, and then calls CategoryFreeList to free the list items array. Your application is responsible for checking the value of categoryP upon return and updating the display or changing the record’s category to the new selection.

**Compatibility**

Implemented only if 2.0 New Feature Set is present.

The constants categoryDefaultEditCategoryString and categoryHideEditCategory are defined only if 3.5 New Feature Set is present. In earlier versions, you can suppress the Edit Categories string by passing 0 for the editingStrID parameter, or include the item by passing categoryEditStrID.

**See Also**

See Also CategorySelectV10
CategorySelectV10

**Purpose**  Process the selection and editing of categories.

**Declared In**  Category.h

**Prototype**  
Boolean CategorySelectV10 (DmOpenRef db, 
const FormType *frm, UInt16 ctlID, UInt16 lstID, 
Boolean title, UInt16 *categoryP, 
Char *categoryName)

**Parameters**

- -> db  Open database containing the categories.
- -> frm  Form that contains the category popup list.
- -> ctlID  ID of the popup trigger.
- -> lstID  ID of the popup list.
- -> title  true to have an “All” list item. (In general, if the trigger is in the form’s title bar, it should have an “All” item. If the trigger is elsewhere in the form, it should not.)
Categories
Category Functions

<-> categoryIndex
Index of the selected category. The index is the index into the categoryLabels array.

<-> categoryName
Name of the selected category.

Result
Returns true if any of the following conditions are true:
- The current category is renamed.
- The current category is deleted.
- The current category is merged with another category.

Compatibility
This function corresponds to the Palm OS 1.0 version of CategorySelect. It is obsolete.

CategorySetName

Purpose
Change the category name in the AppInfoType structure, or delete a category.

Declared In
Category.h

Prototype
void CategorySetName (DmOpenRef db, UInt16 index, const Char *nameP)

Parameters
-> db
Open database containing the category to change.

-> index
Index of category to rename.

-> nameP
The new category name (null-terminated), or NULL to delete the category.

Result
Returns nothing.

Comments
The CategoryEdit function calls this function when a user creates a new category or renames an existing category in the Edit Categories dialog. Your application does not have to call it directly.

Compatibility
Implemented only if 2.0 New Feature Set is present.
**CategorySetTriggerLabel**

**Purpose**
Set the label displayed by the category popup trigger.

**Declared In**
Category.h

**Prototype**
void CategorySetTriggerLabel (ControlType *ctl, Char *name)

**Parameters**
- `<-> ctl`: Pointer to control object (popup trigger) to relabel.
- `<-> name`: Pointer to the name of the new category.

**Result**
Returns nothing.

**Comments**
The `CategorySetTriggerLabel` function calls the `CategoryTruncateName` function to truncate the category name to the maximum length. The maximum length varies, depending upon which ROM is installed in the device.

**NOTE:** This function passes the `name` parameter to the `CategoryTruncateName` function, which means that the `name` value must be modifiable. `CategorySetTriggerLabel` does not make a copy of the string passed, so you must ensure that the string remains valid until the form is closed.

**See Also**
*CtlSetLabel*
Categories
Category Functions

**CategoryTruncateName**

**Purpose**: Truncate a category name so that it’s short enough to display. The category name is truncated if it’s longer than `maxWidth`.

**Declared In**: Category.h

**Prototype**: `void CategoryTruncateName (Char *name, UInt16 maxWidth)`

**Parameters**

- `<-> name` Category name to truncate. Upon return, contains the truncated name.
- `-> maxWidth` Maximum size, in pixels, of truncated category (including ellipsis).

**Result**: Returns nothing.
Clipboard

This chapter provides reference material for the clipboard API defined in Clipboard.h. It covers:

- Clipboard Data Structures
- Clipboard Functions

Clipboard Data Structures

ClipboardFormatType

The ClipboardFormatType enum specifies the type of data to add to the clipboard or retrieve from the clipboard.

```c
enum clipboardFormats {
    clipboardText,
    clipboardInk,
    clipboardBitmap
};
typedef enum clipboardFormats ClipboardFormatType;
```

Value Descriptions

- clipboardText: Textual data. This is the most commonly used clipboard.
- clipboardInk: Reserved.
- clipboardBitmap: Bitmap data.

Clipboards for each type of data are separately maintained. That is, if you add a string of text to the clipboard, then add a bitmap, then ask to retrieve a clipboardText item from the clipboard, you will receive the string you added before the bitmap; the bitmap does not overwrite textual data and vice versa.
Clipboard Functions

**ClipboardAddItem**

**Purpose** Add the item passed to the specified clipboard. Replaces the current item (if any) of that type.

**Declared In** Clipboard.h

**Prototype**
```c
void ClipboardAddItem
(const ClipboardFormatType format,
 const void *ptr, UInt16 length)
```

**Parameters**
- `format` Text, ink, bitmap, etc. See `ClipboardFormatType`.
- `ptr` Pointer to the item to place on the clipboard.
- `length` Size in bytes of the item to place on the clipboard.

**Result** Returns nothing.

**Comments** The clipboard makes a copy of the data that you pass to this function. Thus, you may free any data that you’ve passed to the clipboard without destroying the contents of the clipboard. You may also add constant data or stack-based data to the clipboard.

**WARNING!** You can’t add null-terminated strings to the clipboard.

**See Also** FldCut, FldCopy
**ClipboardAppendItem**

**Purpose**  
Append data to the item on the clipboard.

**Declared In**  
Clipboard.h

**Prototype**  
Err ClipboardAppendItem  
(const ClipboardFormatType format,  
const void *ptr, UInt16 length)

**Parameters**

- **format**  
Text, ink, bitmap, etc. See ClipboardFormatType. This function is intended to be used only for the clipboardText format.

- **ptr**  
Pointer to the data to append to the item on the clipboard.

- **length**  
Size in bytes of the data to append to the clipboard.

**Result**  
0 upon success or memErrNotEnoughSpace if there is not enough space to append the data to the clipboard.

**Comments**

This function differs from ClipboardAddItem in that it does not overwrite data already on the clipboard. It allows you to create a large text item on the clipboard from several small disjointed pieces. When other applications retrieve the text from the clipboard, it’s retrieved as a single unit.

This function simply appends the specified item to the item already on the clipboard without attempting to parse the format. It’s assumed that you’ll call it several times over a relatively short interval and that no other application will attempt to retrieve text from the clipboard before your application is finished appending.

**Compatibility**  
Implemented only if 3.2 New Feature Set is present.
Clipboard
Clipboard Functions

**ClipboardGetItem**

**Purpose**  
Return the handle of the contents of the clipboard of a specified type and the length of a clipboard item.

**Declared In**  
Clipboard.h

**Prototype**  
MemHandle ClipboardGetItem
(const ClipboardFormatType format, UInt16 *length)

**Parameters**

- **format**  
Text, ink, bitmap, etc. See *ClipboardFormatType*.

- **length**  
The length in bytes of the clipboard item is returned here.

**Result**  
Handle of the clipboard item.

**Comments**  
The handle returned is a handle to the actual clipboard chunk. It is not suitable for passing to any API that modifies memory (such as *FldSetTextHandle*). Consider this to be read-only access to the chunk. Copy the contents of the clipboard to your application’s own storage as soon as possible and use that reference instead of the handle returned by this function.

Don’t free the handle returned by this function; it is freed when a new item is added to the clipboard.

Text retrieved from the clipboard does not have a null terminator. You must use the length parameter to determine the length in bytes of the string you’ve retrieved.
Controls

This chapter describes the control object API as declared in the header file Control.h. It discusses the following topics:

- Control Data Structures
- Control Resources
- Control Functions

For more information on controls, see the section “Offscreen Windows” in the Palm OS Programmer’s Companion, vol. I.

Control Data Structures

**ButtonFrameType**

The ButtonFrameType enum specifies the border style for the button. It defines values for the frame field of ControlAttrType.

```c
enum buttonFrames {noButtonFrame,
    standardButtonFrame, boldButtonFrame,
    rectangleButtonFrame};
typedef enum buttonFrames ButtonFrameType;
```

**Value Descriptions**

- **noButtonFrame**
  - The button has no border.

- **standardButtonFrame**
  - Standard button rectangular border with rounded corners.

- **boldButtonFrame**
  - Bolded rectangular border with rounded corners.

- **rectangleButtonFrame**
  - Rectangular border with square corners.
ControlAttrType

The ControlAttrType bit field specifies the control’s visible characteristics. It is defined as follows:

```c
typedef struct  {
    UInt8 usable     :1;
    UInt8 enabled    :1;
    UInt8 visible    :1;
    UInt8 on         :1;
    UInt8 leftAnchor :1;
    UInt8 frame      :3;
    UInt8 drawnAsSelected : 1;
    UInt8 graphical  :1;
    UInt8 vertical   :1;
} ControlAttrType;
```

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the ControlAttrType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Your code should treat the ControlAttrType structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

**Field Descriptions**

- **usable**
  
  If 0, the control is not considered to be part of the interface of the current application, and it doesn’t appear on screen. You can use `CtlSetUsable`, `CtlShowControl`, or `CtlHideControl` to set or clear this value.

- **enabled**
  
  If 0, the control is visible but doesn’t respond to the pen. This value is set by `CtlSetEnabled` and returned by `CtlEnabled`. 
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>visible</td>
<td>Set and cleared internally when the control is drawn (<a href="#">CtlDrawControl</a>) and erased (<a href="#">CtlEraseControl</a>).</td>
</tr>
<tr>
<td>on</td>
<td>If set, the control has the value “on.” For example, a check box that has the on value has a check mark displayed in it. Use <a href="#">CtlGetValue</a> and <a href="#">CtlSetValue</a> to retrieve and set this value.</td>
</tr>
<tr>
<td>leftAnchor</td>
<td>Used by controls that expand and shrink their width when the label is changed. If this attribute is set, the left bound of the control is fixed.</td>
</tr>
<tr>
<td>frame</td>
<td>The type of frame drawn around the button controls. See <a href="#">ButtonFrameType</a> for possible values. Only button controls use this attribute; for all other controls, the <a href="#">ControlStyleType</a> determines the frame.</td>
</tr>
<tr>
<td>drawnAsSelected</td>
<td>Used on Palm OS® release 3.5 for button controls that contain no text (indicating that the button is displayed on top of a bitmap). If set, the button is drawn as inverted. If clear, the button is drawn normally.</td>
</tr>
<tr>
<td>graphical</td>
<td>If set, the control is a graphical control, slider, or feedback slider.</td>
</tr>
<tr>
<td>vertical</td>
<td>Not currently used.</td>
</tr>
</tbody>
</table>

**Compatibility**
The `drawnAsSelected`, `graphical`, and `vertical` attributes are only present if [3.5 New Feature Set](#) is present.

**ControlPtr**
The `ControlPtr` is a pointer to a [ControlType](#) structure.
typedef ControlType* ControlPtr;

ControlStyleType
The ControlStyleType enum specifies values for the ControlType style field, which specifies the type of the control (button, push button, and so on).

enum controlStyles {buttonCtl, pushButtonCtl, checkboxCtl, popupTriggerCtl, selectorTriggerCtl, repeatingButtonCtl, sliderCtl, feedbackSliderCtl};
typedef enum controlStyles ControlStyleType;

Value Descriptions
buttonCtl Button. Buttons display a text label in a box. The ButtonFrameType specifies the type of box.
pushButtonCtl Push button. Selecting a push button inverts its display so that it appears highlighted.
checkboxCtl Check box. Check boxes display a setting of either on (checked) or off (unchecked)
popupTriggerCtl Popup trigger. Popup triggers display a graphic element followed by a text label. They are used to display popup lists.
selectorTriggerCtl Selector trigger. Selector triggers display a text label surrounded by a gray rectangular frame. The control expands or contracts to the width of the new label.
repeatingButtonCtl Repeating button. Repeating buttons look like buttons; however, a repeating button is repeatedly selected if the user holds the pen on it.
sliderCtl

Slider. Sliders display two bitmaps: one representing the current value (the thumb), and another representing the scale of available values. The user can slide the thumb to the left or the right to change the value.

feedbackSliderCtl

Feedback slider. A feedback slider looks like a slider; however, a feedback slider sends events each time the thumb moves while the pen is still down. A regular slider sends an event only when the user releases the pen.

Compatibility

The sliderCtl and feedbackSliderCtl values are only defined if 3.5 New Feature Set is present.

ControlType

The ControlType structure defines the type and characteristics of a control. It is defined as follows:

```
typedef struct {
   UInt16 id;
   RectangleType bounds;
   Char * text;
   ControlAttrType attr;
   ControlStyleType style;
   FontID font;
   UInt8 group;
   UInt8 reserved;
} ControlType;
```

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the ControlType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.
Controls
Control Data Structures

Your code should treat the ControlType structure as opaque. The fields in the struct are set by values you specify when you create the control resource, and they typically do not change. Use the functions specified in the descriptions below to retrieve and set the values. Do not attempt to change structure member values directly.

Field Descriptions

id       ID value you specified when you created the control resource.

bounds   Bounds of the control, in window-relative coordinates. The control’s text label is clipped to the control’s bounds. The control’s frame is drawn around (outside) the bounds of the control. FrmGetObjectBounds and FrmSetObjectBounds retrieve and set this value.

text     Pointer to the control’s label. If text is NULL, the control has no label. Use CtlGetLabel and CtlSetLabel to retrieve and set this value.

attr      Control attributes. See ControlAttrType.

style    Style of the control. See ControlStyleType.

font     Font to use to draw the control’s label.

group    Group ID of a push button or a check box that is part of an exclusive group. The control routines don’t automatically turn one control off when another is selected. It’s up to the application or a higher-level object, like a dialog box, to manage this.

reserved Reserved for future use.

GraphicControlType

The GraphicControlType struct defines a graphical control. A graphical control is like any other control except that it displays a bitmap in place of the text label.
typedef struct GraphicControlType {
    UInt16             id;
    RectangleType      bounds;
    DmResID            bitmapID;
    DmResID            selectedBitmapID;
    ControlAttrType    attr;
    ControlStyleType   style;
    FontID             unused;
    UInt8              group;
    UInt8              reserved;
} GraphicControlType;

WARNING!  PalmSource, Inc. does not support or provide backward compatibility for the GraphicControlType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Your code should treat the GraphicControlType structure as opaque. The fields in the struct are set by values you specify when you create the control resource, and they typically do not change. Use the functions specified in the descriptions below to retrieve and set the values. Do not attempt to change structure member values directly.

Field Descriptions

id          ID value you specified when you created the control resource.

bounds      Bounds of the control, in window-relative coordinates. The control’s frame is drawn around (outside) the bounds of the control. FrmGetObjectBounds and FrmSetObjectBounds retrieve and set this value.

bitmapID    Resource ID of the bitmap to display in the button. You can use CtlSetGraphics to change this value.
### SliderControlType

The **SliderControlType** struct defines a slider control or a feedback slider control.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>selectedBitmapID</td>
<td>If the button should show a different bitmap when selected, this field contains the resource ID of that bitmap. You typically use this field for push buttons or repeating buttons. <code>CtlSetGraphics</code> can change this value.</td>
</tr>
<tr>
<td>attr</td>
<td>Control attributes. See <a href="#">ControlAttrType</a>. For a graphical control, the graphical attribute must be set. The APIs described in the ControlAttrType section can be used to access the bitfields here. Because the ControlAttrType APIs take a <code>ControlType*</code> as an argument, the <code>GraphicControlType*</code> should be cast to a <code>ControlType*</code> when making the API calls.</td>
</tr>
<tr>
<td>style</td>
<td>Style of the control. See <a href="#">ControlStyleType</a>. A graphical control can be any type of control other than checkboxCtl.</td>
</tr>
<tr>
<td>unused</td>
<td>Unused.</td>
</tr>
<tr>
<td>group</td>
<td>Group ID of a push button that is part of an exclusive group. The control routines don’t automatically turn one control off when another is selected. It’s up to the application or a higher-level object, like a dialog box, to manage this.</td>
</tr>
<tr>
<td>reserved</td>
<td>Reserved for future use.</td>
</tr>
</tbody>
</table>

**Compatibility**

This struct is defined only if [3.5 New Feature Set](#) is present.
typedef struct SliderControlType {
    UInt16               id;
    RectangleType        bounds;
    DmResID              thumbID;
    DmResID              backgroundID;
    ControlAttrType      attr;
    ControlStyleType     style;
    UInt8                reserved;
    Int16                minValue;
    Int16                maxValue;
    Int16                pageSize;
    Int16                value;
    MemPtr               activeSliderP;
} SliderControlType;

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the SliderControlType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Your code should treat the SliderControlType structure as opaque. The fields in the struct are set by values you specify when you create the control resource, and they typically do not change. You can use CtlSetSliderValues to set new minimum, maximum, page size, and current values, and CtlGetSliderValues to retrieve these values. Do not attempt to change structure member values directly.

Field Descriptions

id
ID value you specified when you created the control resource.

bounds
Bounds of the control, in window-relative coordinates. FrmGetObjectBounds and FrmSetObjectBounds retrieve and set this value.
### Controls

**Control Data Structures**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>thumbID</td>
<td>Resource ID of the bitmap to use for the slider knob (called the “thumb”). If NULL, the default bitmap is used.</td>
</tr>
<tr>
<td>backgroundID</td>
<td>Resource ID of the bitmap to use for the slider background. If NULL, the default bitmap is used.</td>
</tr>
<tr>
<td>attr</td>
<td>Control attributes. See <a href="#">ControlAttrType</a>. For a slider, the graphical attribute is set. The APIs described in the ControlAttrType section can be used to access the bitfields here. Because the ControlAttrType APIs take a ControlType* as an argument, the SliderControlType* should be cast to a ControlType* when making the API calls.</td>
</tr>
<tr>
<td>style</td>
<td>Style of the control. See <a href="#">ControlStyleType</a>. Must be sliderCtl or feedbackSliderCtl.</td>
</tr>
<tr>
<td>reserved</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>minValue</td>
<td>Value of the slider when the thumb is all the way to the left.</td>
</tr>
<tr>
<td>maxValue</td>
<td>Value of the slider when the thumb is all the way to the right.</td>
</tr>
<tr>
<td>pageSize</td>
<td>Amount by which to increase or decrease the slider value when the user taps to the right or left of the thumb.</td>
</tr>
<tr>
<td>value</td>
<td>Current value represented by the slider. Use <a href="#">CtlGetValue</a> and <a href="#">CtlSetValue</a> to retrieve and set this value.</td>
</tr>
<tr>
<td>activeSliderP</td>
<td>Pointer to a memory location used when the slider is active. A slider is active if it is currently being drawn or if it is tracking the pen. If the slider is inactive, this pointer is NULL.</td>
</tr>
</tbody>
</table>

**Compatibility**  
This struct is defined only if [3.5 New Feature Set](#) is present.
Control Resources

Different resources are associated with different controls, as follows:

- Button—Button Resource (tBTN)
- Popup trigger—Popup Trigger Resource (tPUT)
- Selector trigger—Selector Trigger Resource (tSLT)
- Repeat control—Repeating Button Resource (tREP)
- Push button—Push Button Resource (tPBN)
- Check box—Check Box Resource (tCBX)
- Slider—Slider Resource (tsld)
- Feedback slider—Feedback Slider Resource (tslf)

Control Functions

CtlDrawControl

**Purpose**
Draw a control object (and the text or graphic in it) on screen.

**Declared In**
Control.h

**Prototype**
void CtlDrawControl (ControlType *controlP)

**Parameters**
- controlP
  Pointer to the control object to draw. (See ControlType.)

**Result**
Returns nothing.

**Comments**
The control is drawn only if its usable attribute is true. This function sets the visible attribute to true.

**Compatibility**
In releases prior to Palm OS® 3.5, it is common to create graphical buttons by drawing a button with no text label on top of a bitmap. If 3.5 New Feature Set is present, you should use graphical controls instead. (See GraphicControlType.) CtlDrawControl attempts
to provide backward compatibility for the old-style graphical buttons.

See Also  CtlSetUsable, CtlShowControl

CtlEnabled

Purpose  Return true if the control responds to the pen.

Declared In  Control.h

Prototype  Boolean CtlEnabled (const ControlType *controlP)

Parameters  -> controlP  Pointer to control object. (See ControlType.)

Result  Returns true if the controls object responds to the pen; false if not.

Comments  This function provides no indication of whether the control is visible on the screen. A control that doesn’t respond to the pen may be visible, and if so, its appearance is no different from controls that do respond to the pen. You might use such a control to display some state of your application that cannot be modified.

See Also  CtlSetEnabled

CtlEraseControl

Purpose  Erase a usable and visible control object and its frame from the screen.

Declared In  Control.h

Prototype  void CtlEraseControl (ControlType *controlP)

Parameters  -> controlP  Pointer to control object to erase. (See ControlType.)
Comments
This function sets the visible attribute to false. If 3.5 New Feature Set is present, it also sets the drawnAsSelected attribute to false.

Don’t call this function directly; instead, use FrmHideObject, which calls this function.

CtlGetLabel

Purpose
Return a character pointer to a control’s text label.

Declared In
Control.h

Prototype
const Char *CtlGetLabel (const ControlType *controlP)

Parameters
-> controlP Pointer to control object. (See ControlType.)

Result
Returns a pointer to a null-terminated string.

Comments
Make sure that controlP is not a graphical control or a slider control. The graphical control and slider control structures do not contain a text label field.

See Also
CtlSetLabel

CtlGetSliderValues

Purpose
Return current values used by a slider control.

Declared In
Control.h

Prototype
void CtlGetSliderValues (const ControlType *ctlP, UInt16 *minValueP, UInt16 *maxValueP, UInt16 *pageSizeP, UInt16 *valueP)

Parameters
-> ctlP Pointer to a control object. (See ControlType.)
**CtlGetValue**

**Purpose**
Return the current value of the specified control.

**Declared In**
Control.h

**Prototype**
Int16 CtlGetValue (const ControlType *controlP)

**Parameters**
-> controlP  
Pointer to a control object. (See ControlType.)

**Result**
Returns the current value of the control. For most controls the return value is either 0 (off) or 1 (on). For sliders, this function returns the value of the value field.

**See Also**
CtlSetValue, FrmGetControlGroupSelection, FrmSetControlGroupSelection, FrmGetControlValue, FrmSetControlValue
CtlHandleEvent

**Purpose**
Handle event in the specified control object.

**Declared In**
Control.h

**Prototype**
Boolean CtlHandleEvent (ControlType *controlP, EventType *pEvent)

**Parameters**
-> controlP Pointer to control object. (See ControlType.)
-> pEvent Pointer to an EventType structure.

**Result**
Returns true if an event is handled by this function. Events that are handled are:

- **penDownEvent** — If the pen is within the bounds of the control
- **ctlEnterEvent, ctlRepeatEvent, and ctlExitEvent** — If the control ID in the event data matches the control’s ID.

**Comments**
The control object must be usable, visible, and respond to the pen for this function to handle the event.

When this routine receives a penDownEvent, it checks if the pen position is within the bounds of the control object. If it is, a ctlEnterEvent is added to the event queue and the routine exits.

When this routine receives a ctlEnterEvent, the control object is redrawn as necessary as either selected or deselected, depending on its previous state.

When this routine receives a ctlEnterEvent or ctlRepeatEvent, it checks that the control ID in the passed event record matches the ID of the specified control. If they match, this routine tracks the pen until it comes up or until it leaves the object’s bounds. When that happens, ctlSelectEvent is sent to the event queue if the pen came up in the bounds of the control. If the pen exits the bounds, a ctlExitEvent is sent to the event queue.
### CtlHideControl

**Purpose**
Set a control’s usable attribute to false and erase the control from the screen.

**Declared In**
Control.h

**Prototype**
```c
void CtlHideControl (ControlType *controlP)
```

**Parameters**
- `-> controlP` Pointer to the control object to hide. (See `ControlType`.)

**Result**
Returns nothing.

**Comments**
A control that is not usable doesn’t draw and doesn’t respond to the pen.

This function is the same as `CtlEraseControl` except that it also sets usable to false (in addition to setting visible to false).

Don’t call this function directly; instead, use `FrmHideObject`, which performs the same function and works for all user interface objects.

**See Also**
- `CtlShowControl`

### CtlHitControl

**Purpose**
Simulate tapping a control. This function adds a `ctlSelectEvent` to the event queue.

**Declared In**
Control.h

**Prototype**
```c
void CtlHitControl (const ControlType *controlP)
```

**Parameters**
- `-> controlP` Pointer to a control object. (See `ControlType`.)

**Result**
Returns nothing.
Comments  Useful for testing.

**CtlNewControl**

**Purpose**  Create a new control object dynamically and install it in the specified form.

**Declared In**  Control.h

**Prototype**
```
ControlType *CtlNewControl (void **formPP,
   UInt16 ID, ControlStyleType style,
   const Char *textP, Coord x, Coord y, Coord width,
   Coord height, FontID font, UInt8 group,
   Boolean leftAnchor)
```

**Parameters**
- `<-> formPP`  Pointer to the pointer to the form in which the new control is installed. This value is not a handle; that is, the formPP value may change if the object moves in memory. In subsequent calls, always use the new formPP value returned by this function.
- `-> ID`  Symbolic ID of the control.
- `-> style`  A `ControlStyleType` value specifying the kind of control to create: button, push button, repeating button, check box, popup trigger, or popup selector. To create a graphical control or slider control dynamically, use `CtlNewGraphicControl` or `CtlNewSliderControl`, respectively.
### Controls

#### Control Functions

- **textP**
  Pointer to the control’s label text. If `textP` is `NULL`, the control has no label. Only buttons, push buttons, and text boxes have text labels. Because the contents of this pointer are copied into their own buffer, you can free the `textP` pointer any time after the `CtlNewControl` function returns. The buffer into which this string is copied is freed automatically when you remove the control from the form or delete the form.

- **x**
  Horizontal coordinate of the upper-left corner of the control’s boundaries, relative to the window in which it appears.

- **y**
  Vertical coordinate of the upper-left corner of the control’s boundaries, relative to the window in which it appears.

- **width**
  Width of the control, expressed in pixels. Valid values are 1–160. If the value of either of the `width` or `height` parameters is 0, the control is sized automatically as necessary to display the text passed as the value of the `text` parameter.

- **height**
  Height of the control, expressed in pixels. Valid values are 1–160. If the value of either of the `width` or `height` parameters is 0, the control is sized automatically as necessary to display the text passed as the value of the `text` parameter.

- **font**
  Font used to draw the control’s label.

- **group**
  Group ID of a push button or a check box that is part of an exclusive group. The control routines don’t turn one control off automatically when another is selected. It’s up to the application or a higher-level object, such as a dialog box, to manage this.
Controls
Control Functions

-> leftAnchor true specifies that the left bound of this control is fixed. This attribute is used by controls that resize dynamically in response to label text changes.

Result Returns a pointer to the new control.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also CtlValidatePointer, FrmRemoveObject

CtlNewGraphicControl

Purpose Create a new graphical control dynamically and install it in the specified form.

Declared In Control.h

Prototype GraphicControlType *CtlNewGraphicControl
(void **formPP, UInt16 ID,
ControlStyleType style, DmResID bitmapID,
DmResID selectedBitmapID, Coord x, Coord y,
Coord width, Coord height, UInt8 group,
Boolean leftAnchor)

Parameters <-> formPP Pointer to the pointer to the form in which the new control is installed. This value is not a handle; that is, the formPP value may change if the object moves in memory. In subsequent calls, always use the new formPP value returned by this function.

-> ID Symbolic ID of the control.

-> style A ControlStyleType value specifying the kind of control to create: button, push button, popup trigger, repeating button, or popup selector. Graphic controls cannot be check boxes.
Controls

Control Functions

- `bitmapID` Resource ID of the bitmap to display on the control.

- `selectedBitmapID` Resource ID of the bitmap to display when the control is selected, if different from `bitmapID`.

- `x` Horizontal coordinate of the upper-left corner of the control’s boundaries, relative to the window in which it appears.

- `y` Vertical coordinate of the upper-left corner of the control’s boundaries, relative to the window in which it appears.

- `width` Width of the control, expressed in pixels. Valid values are 1–160.

- `height` Height of the control, expressed in pixels. Valid values are 1–160.

- `group` Group ID of a push button that is part of an exclusive group. The control routines don’t turn one control off automatically when another is selected. It’s up to the application or a higher-level object, such as a dialog box, to manage this.

- `leftAnchor` `true` specifies that the left bound of this control is fixed.

Result Returns a pointer to the new graphical control. See `GraphicControlType`.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `CtlNewSliderControl`, `CtlNewControl`, `CtlValidatePointer`, `FrmRemoveObject`
CtlNewSliderControl

Purpose  Create a new slider or feedback slider dynamically and install it in the specified form.

Declared In  Control.h

Prototype  SliderControlType *CtlNewSliderControl
(void **formPP, UInt16 ID,
ControlStyleType style, DmResID thumbID,
DmResID backgroundID, Coord x, Coord y,
Coord width, Coord height, UInt16 minValue,
UInt16 maxValue, UInt16 pageSize, UInt16 value)

Parameters  <-> formPP  Pointer to the pointer to the form in which the new control is installed. This value is not a handle; that is, the formPP value may change if the object moves in memory. In subsequent calls, always use the new formPP value returned by this function.

-> ID  Symbolic ID of the slider.

-> style  Either sliderCtl or feedbackSliderCtl. See ControlStyleType.

-> thumbID  Resource ID of the bitmap to display as the slider thumb. The slider thumb is the knob that the user can drag to change the slider’s value. To use the default thumb bitmap, pass NULL for this parameter.

-> backgroundID  Resource ID of the bitmap to display as the slider background. To use the default background bitmap, pass NULL for this parameter.

-> x  Horizontal coordinate of the upper-left corner of the slider’s boundaries, relative to the window in which it appears.
Controls
Control Functions

- \( y \)  
  Vertical coordinate of the upper-left corner of the slider’s boundaries, relative to the window in which it appears.

- \( \text{width} \)  
  Width of the slider, expressed in pixels. Valid values are 1–160.

- \( \text{height} \)  
  Height of the slider, expressed in pixels. Valid values are 1–160.

- \( \text{minValue} \)  
  Value of the slider when its thumb is all the way to the left.

- \( \text{maxValue} \)  
  Value of the slider when its thumb is all the way to the right.

- \( \text{pageSize} \)  
  Amount by which to increase or decrease the slider’s value when the user clicks to the right or left of the thumb.

- \( \text{value} \)  
  The initial value to display in the slider.

Result
Returns a pointer to the new slider control. See SliderControlType.

Compatibility
Implemented only if 3.5 New Feature Set is present.

See Also
CtlNewGraphicControl, CtlNewControl, CtlValidatePointer, FrmRemoveObject

CtlSetEnabled

Purpose
Set a control as enabled or disabled. Disabled controls do not respond to the pen.

Declared In
Control.h

Prototype
void CtlSetEnabled (ControlType *controlP, Boolean usable)

Parameters
- \( \text{controlP} \)  
  Pointer to a control object. (See ControlType.)
Controls
Control Functions

-> usable  true to enable the control; false to disable the control.

Result  Returns nothing.

Comments  If you disable a visible control, the control is still displayed, and its appearance is no different from controls that do respond to the pen. You might use such a control to inform your users of some state of your application that cannot be modified.

See Also  CtlEnabled

CtlSetGraphics

Purpose  Set the bitmaps for a graphical control and redraw the control if it is visible.

Declared In  Control.h

Prototype  void CtlSetGraphics (ControlType *ctlP, DmResID newBitmapID, DmResID newSelectedBitmapID)

Parameters  -> ctlP  Pointer to a graphical control object. (See GraphicControlType.)

-> newBitmapID  Resource ID of a new bitmap to display on the control, or NULL to use the current bitmap.

-> newSelectedBitmapID  Resource ID of a new bitmap to display when the control is selected, or NULL to use the current selected bitmap.

Result  Returns nothing.

Comments  If ctlP is not a graphical control, this function immediately returns.
**Compatibility**
Implemented only if 3.5 New Feature Set is present.

**See Also**
GraphicControlType

---

### CtlSetLabel

**Purpose**
Set the current label for the specified control object and redraw the control if it is visible.

**Declared In**
Control.h

**Prototype**
```c
void CtlSetLabel (ControlType *controlP,
                 const Char *newLabel)
```

**Parameters**
- `-> controlP` Pointer to a control object. (See ControlType.)
- `-> newLabel` Pointer to the new text label. Must be a null-terminated string.

**Result**
Returns nothing.

**Comments**
This function resizes the width of the control to the size of the new label.

This function stores the newLabel pointer in the control’s data structure. It doesn’t make a copy of the string that is passed in. Therefore, if you use CtlSetLabel, you must manage the string yourself. You must ensure that it persists for as long as it is being displayed (that is, for as long as the control is displayed or until you call CtlSetLabel with a new string), and you must free the string after it is no longer in use (typically after the form containing the control is freed).

If you never use CtlSetLabel, you do not need to worry about freeing a control’s label.

Make sure that controlP is not a graphical control or a slider control. The graphical controls and slider control structures do not...
contain a text label field, so attempting to set one will crash your application.

See Also  
CtlGetLabel

CtlSetSliderValues

Purpose  
Change a slider control’s values and redraw the slider if it is visible.

Declared In  
Control.h

Prototype  
void CtlSetSliderValues (ControlType *ctlP,
const UInt16 *minValueP, const UInt16 *maxValueP,
const UInt16 *pageSizeP, const UInt16 *valueP)

Parameters  
-> ctlP  
Pointer to an inactive slider or feedback slider control. (See SliderControlType.)

-> minValueP  
Pointer to a new value to use for the slider’s minimum or NULL if you don’t want to change this value.

-> maxValueP  
Pointer to a new value to use for the slider’s maximum, or NULL if you don’t want to change this value.

-> pageSizeP  
Pointer to a new value to use for the slider’s page size, or NULL if you don’t want to change this value.

-> valueP  
Pointer to a new value to use for the current value, or NULL if you don’t want to change this value.

Result  
Returns nothing.

Comments  
The control’s style must be sliderCtl or feedbackSliderCtl, and it not be currently tracking the pen. If the slider is currently tracking the pen, use CtlSetValue to set the value field.
Controls
Control Functions

Compatibility
Implemented only if 3.5 New Feature Set is present.

See Also
CtlGetSliderValues, SliderControlType

CtlSetUsable

Purpose
Set a control to usable or not usable.

Declared In
Control.h

Prototype
gvoid CtlSetUsable (ControlType *controlP, Boolean usable)

Parameters
-> controlP Pointer to a control object. (See ControlType.)
-> usable true to have the control be usable; false to have the control be not usable.

Result
Returns nothing.

Comments
A control that is not usable doesn’t draw and doesn’t respond to the pen.
This function doesn’t usually update the control.

See Also
CtlEraseControl, CtlHideControl, CtlShowControl

CtlSetValue

Purpose
Set the current value of the specified control. If the control is visible, it’s redrawn.

Declared In
Control.h

Prototype
gvoid CtlSetValue (ControlType *controlP, Int16 newValue)

Parameters
-> controlP Pointer to a control object. (See ControlType.)
-> newValue  
New value to set for the control. For sliders, specify a value between the slider’s minimum and maximum. For graphical controls, push buttons, or check boxes, specify 0 for off, nonzero for on.

Result  
Returns nothing.

Comments  
This function works only with graphical controls, sliders, push buttons, and check boxes. If you set the value of any other type of control, the behavior is undefined.

Compatibility  
Sliders and graphical controls are only supported if 3.5 New Feature Set is present.

See Also  
CtlGetValue, FrmGetControlGroupSelection, FrmSetControlGroupSelection, FrmGetControlValue, FrmSetControlValue

CtlShowControl

Purpose  
Set a control’s usable attribute to true and draw the control on the screen. This function calls CtlDrawControl.

Declared In  
Control.h

Prototype  
void CtlShowControl (ControlType *controlP)

Parameters  
-> controlP  
Pointer to a control object. (See ControlType.)

Result  
Returns nothing.

Comments  
If the control is already usable, this function is the functional equivalent of CtlDrawControl. Sets the visible and the usable attributes to true. (See ControlAttrType.)
Don’t use this function directly; instead use `FrmShowObject`, which does the same thing.

**See Also**  
`CtlHideControl`

---

### CtlValidatePointer

**Purpose**
Returns `true` if the specified pointer references a valid control object.

**Declared In**
`Control.h`

**Prototype**
```c
Boolean CtlValidatePointer
    (const ControlType *controlP)
```

**Parameters**
`-> controlP`  
Pointer to a control. (See `ControlType`.)

**Result**
Returns `true` when passed a valid pointer to a control; otherwise, returns `false`.

**Comments**
For debugging purposes; do not include this function in commercial products. In debug builds, this function displays a dialog and waits for the debugger when an error occurs.

**Compatibility**
Implemented only if `3.0 New Feature Set` is present.

**See Also**  
`FrmValidatePtr`, `WinValidateHandle`
Date and Time Selector

The Palm OS® UI provides two system resources for accepting date and time input values. These resources are dialog boxes that contain UI gadgetry for entering dates and times. The Palm OS UI also provides routines to manage the interaction with these resources. This chapter describes those functions.

The API described in this chapter is declared in the header files Day.h, SelDay.h, SelTime.h, and SelTimeZone.h.

Date and Time Selections Data Structures

**SelectDayType**

```c
typedef enum {
    selectDayByDay,     // return d/m/y
    selectDayByWeek,   // return d/m/y with d as
                        // same day of the week
    selectDayByMonth   // return d/m/y with d as
                        // same day of the month
} SelectDayType;
```

**DaySelectorType**

```c
typedef struct DaySelectorType {
    RectangleType  bounds;
    Boolean        visible;
    UInt8          reserved1;
    Int16           visibleMonth; // month actually
                        // displayed
} DaySelectorType;
```
Date and Time Selector

Date and Time Selection Functions

```c
Int16     visibleYear;  // year actually
                      // displayed
DateTimeType selected;
SelectDayType selectDayBy;
UInt8      reserved2;
} DaySelectorType;
```

HMSTime

typedef struct {
  UInt8 hours;
  UInt8 minutes;
  UInt8 seconds;
  UInt8 reserved;
} HMSTime;

Date and Time Selection Functions

DayDrawDays

**Purpose**
Draw only the days-of-the-month portion of a day selector control object.

**Declared In**
Day.h

**Prototype**
```c
void DayDrawDays
  (const DaySelectorType *selectorP)
```

**Parameters**
- selectorP Pointer to the control object to draw.

**Result**
Nothing.

**Comments**
This function is used when the year or month changes. Only drawing the portion of the control that presents the days of the month avoids the flicker that would occur if the week titles were redrawn.
DayDrawDaySelector

Purpose
Draw a day selector control object on screen.

Declared In
Day.h

Prototype
void DayDrawDaySelector
(const DaySelectorType *selectorP)

Parameters
-> selectorP       Pointer to the control object to draw.

Result
Nothing.

Comments
The control is drawn only if it is usable.

Compatibility
If 5.0 New Feature Set is present this function is unimplemented.

See Also
DayDrawDays

DayHandleEvent

Purpose
Handle event in the specified control. This routine handles two types of events, penDownEvent and ctlEnterEvent.

Declared In
Day.h

Prototype
Boolean DayHandleEvent
(DaySelectorType *selectorP,
const EventType *pEvent)

Parameters
-> selectorP       Pointer to control object.
Date and Time Selector
Date and Time Selection Functions

-> pEvent Pointer to an EventType structure.

Result true if the event was handled or false if it was not.

Posts a daySelectEvent with information on whether to use the date.

Comments A date is used if the user selects a day in the visible month.

When this routine receives a penDownEvent, it checks if the pen position is within the bounds of the control object. If it is, a dayEnterEvent is added to the event queue and the routine exits.

When this routine receives a dayEnterEvent, it checks that the control id in the event record matches the id of the control specified. If they match, this routine will track the pen until it comes up in the bounds in which case daySelectEvent is sent.

If the pen exits the bounds a dayExitEvent is sent.

SelectDay

Purpose Display a form showing a date; allow user to select a different date.

Declared In SelDay.h

Prototype Boolean SelectDay
(const SelectDayType selectDayBy, Int16 *month, Int16 *day, Int16 *year, const Char *title)

Parameters selectDayBy The method by which the user should choose the day. Possible values are selectDayByDay, selectDayByWeek, and selectDayByMonth. See SelectDayType

<-> month, day, year Month, day, and year selected.

<-> title String title for the dialog.

Result true if the OK button was pressed. If true, month, day, and year contain the new date.
Compatibility Implemented only if 2.0 New Feature Set is present.

See Also SelectDayV10

SelectDayV10

Purpose Display a form showing a date, allow user to select a different date.

Declared In SelDay.h

Prototype Boolean SelectDayV10 (Int16 *month, Int16 *day, Int16 *year, const Char *title)

Parameters <-> month, day, year
   Month, day, and year selected. The initial values passed in these parameters must be valid.

-> title String title for the dialog.

Result Returns true if the OK button was pressed. In that case, the parameters passed are changed.

Compatibility This function corresponds to the 1.0 version of SelectDay.

See Also SelectDay

SelectOneTime

Purpose Display a form showing the time and allow the user to select a different time.

Declared In SelTime.h

Prototype Boolean SelectOneTime (Int16 *hour, Int16 *minute, const Char *titleP)

Parameters <-> hour The hour selected in the form.
Date and Time Selector
Date and Time Selection Functions

<-> minute
   The minute selected in the form.

-> titleP
   A pointer to a string to display as the title.
   Doesn’t change as the function executes.

Result
Returns true if the user selects OK and false otherwise. If true is returned, the values in hour and minute have probably been changed.

Comments
Use this function instead of SelectTime if you want to display a dialog that specifies a single point in time, not a range of time from start to end.

Compatibility
Implemented only if 3.1 New Feature Set is present.

See Also
SelectTimeV33

SelectTime

Purpose
Display a form showing a start and end time. Allow the user to select a different time.

Declared In
SelTime.h

Prototype
Boolean SelectTime (TimeType *startTimeP, TimeType * EndTimeP, Boolean untimed, const Char *titleP, Int16 startOfDay, Int16 endOfDay, Int16 startOfDay)

Parameters
<-> startTimeP, EndTimeP
   Pointers to values of type TimeType. Pass values to display in these two parameters. If the user makes a selection and taps the OK button, the selected values are returned here.

-> untimed
   Pass in true to indicate that no time is selected. If the user sets the time to no time then startTimeP and EndTimeP are both set to the constant noTime (-1).
-> titleP A pointer to a string to display as the title. Doesn’t change as the function executes.

-> startOfDay The hour that the hour list displays at its top. To see earlier hours, the user can scroll the list up. The value must be between 0 to 12, inclusive.

-> endOfDay The hour used when the “All Day” button is selected.

-> startOfDay The first hour initially visible.

Result Returns true if the user selects OK and false otherwise. If true is returned, the values in hour and minute have probably been changed.

Comments This version of SelectTime adds the endOfDay and startOfDay functionality.

Compatibility Implemented if 3.5 New Feature Set is present.

See Also SelectDay, SelectOneTime

SelectTimeV33

Purpose Display a form showing the time and allow the user to select a different time.

This function is obsolete and should not be used.
Date and Time Selector
Date and Time Selection Functions

Declared In  SelTime.h

Prototype  Boolean SelectTimeV33 (TimeType *startTimeP, TimeType *EndTimeP, Boolean untimed, const Char *titleP, Int16 startOfDay)

Parameters  <-> startTimeP, EndTimeP  
Pointers to values of type TimeType. Pass values to display in these two parameters. If the user makes a selection and taps the OK button, the selected values are returned here.

  -> untimed  
Pass in true to indicate that no time is selected. If the user sets the time to no time then startTimeP and EndTimeP are both set to the constant noTime (-1).

  -> titleP  
A pointer to a string to display as the title. Doesn’t change as the function executes.

  -> startOfDay  
The hour that the hour list displays at its top. To see earlier hours, the user can scroll the list up. The value must be between 0 to 12, inclusive.

Result  Returns true if the user selects OK and false otherwise. If true is returned, the values in hour and minute have probably been changed.

Comments  NOTE: Obsolete functions are provided ONLY for backward compatibility; for example, so a 1.0 application will work on 3.x OS releases. New code should not call these routines!

See Also  SelectDay, SelectOneTime
SelectTimeZone

**Purpose** Display a form that allows the user to select a different time zone.

**Declared In** SelTimeZone.h

**Prototype**

```c
Boolean SelectTimeZone (Int16 *ioTimeZoneP,
LmLocaleType *ioLocaleInTimeZoneP,
const Char *titleP, Boolean showTimes,
Boolean anyLocale)
```

**Parameters**

- **<> ioTimeZoneP**
  A pointer to the time zone, given as minutes east of Greenwich Mean Time (GMT). The initial value is used as the initial selection in the form. Upon return, this parameter contains a pointer to the new time zone that the user selected.

- **<> ioLocaleInTimeZoneP**
  A pointer to a locale (see `LmLocaleType`) that identifies the time zone country. This parameter is used for countries that share a time zone, such as Canada and Chile.

  If the time zone specified by `ioTimeZoneP` is specific to a particular country, you do not have to initialize this parameter. Instead, set the `anyLocale` parameter to `true` to have this parameter ignored upon entry.

- **-> titleP**
  A string to use as the title for the dialog. Pass `NULL` to use the default title, which is “Set Time Zone”.

- **-> showTimes**
  If `true`, the dialog shows the correct times in both the current and newly selected time zones. If `false`, the dialog doesn’t show the current time. Using `false` provides a larger area for the list of time zones.
**Date and Time Selector**

**Date and Time Selection Functions**

-> anyLocale

If true, ignore ioLocaleInTimeZoneP upon entry.

**Result**

Returns true if the user clicked the OK button in the dialog to change the time zone, or false if the user clicked the Cancel button.

**Comments**

The time zones displayed in the form are listed by country. For this reason, if the time zone specified by ioTimeZoneP is shared by several countries, you need to supply a value for ioLocaleTimeZoneP to identify which country should be selected when the list is first displayed. You can use the constant lmAnyLanguage as the value for the language field of the structure pointed to by this parameter.

If you don’t care which value is initially selected, pass true for the anyLocale parameter. In this case, the first country that matches the GMT offset given in ioTimeZoneP is selected.

You might want to use the current time zone stored in the system preferences as the initial value for the ioTimeZoneP parameter. To obtain this time zone, do the following:

```c
Int16 timeZone =
    (Int16)PrefGetPreference(prefTimeZone);
CountryType timeZoneCountry = (CountryType)
    PrefGetPreference(prefTimeZoneCountry);
LmLocaleType timeZoneLocale;
Boolean change =
    SelectTimeZone(&timeZone,
                    &timeZoneLocale, NULL, true, false);
```

**Compatibility**

Implemented if 4.0 New Feature Set is present.
Palm OS Programmer’s API Reference

This chapter provides the following information about field objects:

- Field Data Structures
- Field Resources
- Field Functions

The header file `Field.h` declares the API that this chapter describes. For more information on fields, see the section “Fields” in the *Palm OS Programmer’s Companion*, vol. I.

### Field Data Structures

#### FieldAttrType

The `FieldAttrType` bit field defines the visible characteristics of the field. The functions `FldGetAttributes` and `FldSetAttributes` return and set these values. There are other functions that retrieve or set individual attributes defined here. Those functions are noted below.

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `FieldAttrType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```c
typedef struct {
    Uint16 usable :1;
    Uint16 visible :1;
    Uint16 editable :1;
    Uint16 singleLine :1;
    Uint16 hasFocus :1;
} FieldAttrType;
```
Fields
Field Data Structures

```c
Uint16 dynamicSize :1;
Uint16 insPtVisible :1;
Uint16 dirty :1;
Uint16 underlined :2;
Uint16 justification :2;
Uint16 autoShift :1;
Uint16 hasScrollBar :1;
Uint16 numeric :1;
}
FieldAttrType;
```

Field Descriptions

**usable**
If not set, the field object is not considered part of the current interface of the application, and it doesn’t appear on screen. The function `FldSetUsable` sets this value, but it is better to use `FrmShowObject`.

**visible**
Set or cleared internally when the field object is drawn with `FldDrawField` or `FrmShowObject`, and erased with `FldEraseField` or `FrmHideObject`.

**editable**
If not set, the field object doesn’t accept Graffiti® input or editing commands and the insertion point cannot be positioned with the pen. The text can still be selected and copied.

**singleLine**
If set, the field is a single line of text high and the text does not wrap when it exceeds the width of the field. If not set, the text wraps to fill multiple lines.

**hasFocus**
Set internally when the field has the current focus. The blinking insertion point appears in the field that has the current focus. Use the function `FrmSetFocus` and `FldReleaseFocus` to set this value.
**dynamicSize**  If set, a fldHeightChangedEvent is generated whenever the number of lines needs to increase or decrease. Your application needs to respond to this event by adjusting the size of the field’s bounding box. If not set, the text wraps to fill more (or less) lines as required, but the event is not generated. *Note that this bit does not cause the field to change size automatically;* your application must respond to the fldHeightChangedEvent and resize the field itself.

Set this attribute to false if the Single Line attribute is set.

**insPtVisible**  If set, the insertion point is scrolled into view. This attribute is set and cleared internally.

**dirty**  If set, the user has modified the field. The functions FldDirty and FldSetDirty retrieve this field’s value.

**underlined**  If set each line of the field, including blank lines, is underlined. Possible values are defined by the UnderlineModeType defined in Window.h:

- noUnderline
- grayUnderline
- solidUnderline
- colorUnderline

Editable text fields generally use grayUnderline as the value.

The solidUnderline value is only valid for Palm OS 3.1 and higher.

The colorUnderline value is only valid for Palm OS 3.5 and higher.
**Fields**  
*Field Data Structures*

---

**justification** Specifies the text alignment. Possible values are  
`leftAlign` and `rightAlign`. (left or right justification only;  
`centerAlign` justification is not supported).

**autoShift** If set, Graffiti auto-shift rules are applied.

**hasScrollBar** If set, the field has a scrollbar. The system sends more frequent `fldChangedEvents` so the application can adjust the height appropriately.

**numeric** If set, only the characters 0 through 9 and  
associated separators are allowed to be entered in the field. The associated separators are the  
thousands separator and the decimal character. The values of these two characters depend on  
the settings in the Formats prefs panel.

### FieldPtr

The `FieldPtr` type defines a pointer to a `FieldType` structure.

```c
typedef FieldType *FieldPtr;
```

You pass the `FieldPtr` as an argument to all field functions. You can obtain the `FieldPtr` using the function `FrmGetObjectPtr` in this way:

```c
fldPtr = FrmGetObjectPtr(frm,  
    FrmGetObjectIndex(frm, fldID));
```

where `fldID` is the resource ID assigned when you created the field.

### FieldType

The `FieldType` structure represents a field.
WARNING!  PalmSource, Inc. does not support or provide backward compatibility for the FieldType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```
typedef struct {
  Uint16             id;
  RectangleType      rect;
  FieldAttrType      attr;
  Char               *text;
  MemHandle          textHandle;
  LineInfoPtr        lines;
  Uint16             textLen;
  Uint16             textBlockSize;
  Uint16             maxChars;
  Uint16             selFirstPos;
  Uint16             selLastPos;
  Uint16             insPtXPos;
  Uint16             insPtYPos;
  FontID             fontID;
  Uint8              reserved;
} FieldType;
```

Your code should treat the FieldType structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

**Field Descriptions**

**id**  ID value you specified when you created the field resource. This ID value is included as part of the event data of fldEnterEvent.

**rect**  Position and size of the field object. The functions FldGetBounds, FrmGetObjectBounds, FldSetBounds, and FrmSetObjectBounds retrieve and set this value.
Fields
Field Data Structures

attr
Field object attributes. (See FieldAttrType.)

text
Pointer to the null-terminated string that is displayed by the field object. The functions FldGetTextPtr and FldSetTextPtr retrieve and set this value (see below).

Never set the contents of this string directly; for example, do not pass this pointer as the destination value to a function such as StrCopy.

textHandle
Handle to the stored text or to a database record containing the stored text. The functions FldGetTextHandle and FldSetTextHandle retrieve and set this value.

If textHandle is defined, the field calculates the text pointer when it locks the handle. In general, you should only use FldGetTextPtr and FldSetTextPtr on text fields that aren’t editable. On editable text fields, use FldGetTextHandle and FldSetTextHandle.

Also note that editable text fields allocate the text handle as necessary. If a user starts typing in a field that doesn’t have a text handle allocated, the field will allocate one. The field also resizes the text’s memory block as necessary when the user adds more text.

lines
Pointer to an array of LineInfoType structures. There is one entry in this array for each visible line of the text. (See LineInfoType.) The field code maintains this array internally; you should never change the lines array yourself.

Note that this value is NULL for single line fields, and for fields that do not have an allocated text handle.
**Fields**

**Field Data Structures**

- **textLen**: Length in bytes of the string currently displayed by the field object; the null terminator is excluded. You can retrieve this value with `FldGetTextLength`.

- **textBlockSize**: Allocated size of the memory block that holds the field object’s text string. You can retrieve this value with `FldGetTextAllocatedSize`.

  Fields allocate memory for the field text as needed, several bytes at a time.

  Note that `textBlockSize` may be different from the size of the chunk pointed to by `textHandle`. The `textHandle` may point to a database record that contains, in part, the text displayed by the field. If you called `MemHandleSize` on such a `textHandle`, the number returned may be greater than `textBlockSize`.

- **maxChars**: Maximum number of bytes the field object accepts. The functions `FldGetMaxChars` and `FldSetMaxChars` retrieve and set this value.

  Note the difference between `textLen`, `textBlockSize`, and `maxChars`. `textLen` is the number of bytes of character data that `text` actually holds. `textBlockSize` is the amount of memory currently allocated for the text (which must be greater than or equal to `textLen`), and `maxChars` sets the maximum value that `textBlockSize` and `textLen` can expand to.

  For example, if you’ve created a text field for users to enter their first names in, you might specify that the maximum length of this field is 20 bytes. If a user enters “John” in this field, `textLen` is 4, `textBlockSize` is 16, and `maxChars` is 20.
**Fields**

*Field Data Structures*

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>selFirstPos</td>
<td>Starting character offset in bytes of the current selection. Use <strong>FldGetSelection</strong> and <strong>FldSetSelection</strong> to retrieve and set this value and the selLastPos value.</td>
</tr>
<tr>
<td>selLastPos</td>
<td>Ending character offset in bytes of the current selection. When selFirstPos equals selLastPos, there is no selection.</td>
</tr>
<tr>
<td>insPtXPos</td>
<td>Horizontal location of the insertion point, given as the offset in bytes into the line indicated by insPtYPos. The functions <strong>FldGetInsPtPosition</strong> and <strong>FldSetInsPtPosition</strong> retrieve and set a byte offset calculated from this value. If the insertion point isn't visible—if it's on a line that's either above or below the set of visible lines—insPtXPos is the absolute byte offset of the insertion point.</td>
</tr>
<tr>
<td>insPtYPos</td>
<td>Vertical location of the insertion point, given as the display line where the insertion point is positioned. The first display line is zero. The first display line may be different from the first line of text in the field if the field has been scrolled. The functions <strong>FldGetInsPtPosition</strong> and <strong>FldSetInsPtPosition</strong> retrieve and set a byte offset calculated from this value. If the insertion point isn't visible—if it's on a line that's either above or below the set of visible lines—insPtYPos is set to 0x8000.</td>
</tr>
<tr>
<td>fontID</td>
<td>Font ID for the field. See Font.h for more information. The functions <strong>FldGetFont</strong> and <strong>FldSetFont</strong> retrieve and set this value.</td>
</tr>
<tr>
<td>reserved</td>
<td>Reserved for future use.</td>
</tr>
</tbody>
</table>

**LineInfoPtr**

The **LineInfoPtr** type defines a pointer to the **LineInfoType**.
typedef LineInfoType *LineInfoPtr;

LineInfoType
The LineInfoType structure defines an element in the field’s lines array. The lines array contains the field’s word wrapping information. There is one element in the array per visible line in the field, including visible lines that contain no text. The field code maintains this array internally; you should never change the lines array yourself.

The functions FldCalcFieldHeight, FldGetVisibleLines, FldRecalculateField, and FldGetNumberOfBlankLines retrieve or set information in this structure. The scrolling functions FldGetScrollPosition, FldGetScrollValues, FldScrollField, and FldSetScrollPosition also retrieve or set information in this structure.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the LineInfoType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

typedef struct {
    UInt16       start;
    UInt16       length;
} LineInfoType;

Field Descriptions

start The byte offset into the FieldType’s text field of the first character displayed by this line. If the line is blank, start is equal to textLen and length is 0.

length The length in bytes of the portion of the string displayed on this line. If the line is blank, the length is 0.
Field Resources

The Field Resource (tFLD) represents a field on screen.

Field Functions

**FldCalcFieldHeight**

**Purpose**
Determine the height of a field for a string.

**Declared In**
Field.h

**Prototype**
UInt16 FldCalcFieldHeight (const Char *chars, UInt16 maxWidth)

**Parameters**
- `chars` Pointer to a null-terminated string.
- `maxWidth` Maximum line width in pixels.

**Result**
Returns the total number of lines needed to draw the string passed.

**Comments**
The width of a field is contained in the rect member of the FieldType structure. You can retrieve this value in the following way:

```c
FrmGetObjectBounds(frm, FrmGetObjectIndex(frm, fldID), &myRect);
fieldWidth = myRect.extent.x;
FldCalcFieldHeight(myString, fieldWidth);
```

**See Also**
FldWordWrap
**FldCompactText**

**Purpose**
Compact the memory block that contains the field’s text to release any unused space.

**Declared In**
Field.h

**Prototype**
```c
void FldCompactText (FieldType *fldP)
```

**Parameters**
- `fldP` Pointer to a field object (FieldType structure).

**Result**
Returns nothing.

**Comments**
As characters are added to the field’s text, the block that contains the text is grown. The block is expanded several bytes at a time so that it doesn’t have to expand each time a character is added. This expansion may result in some unused space in the text block. Applications should call this function on field objects that edit data records in place before the field is unlocked, or at any other time when a compact field is desirable; for example, before writing to the storage heap.

**See Also**
FldGetTextAllocatedSize, FldSetTextAllocatedSize

**FldCopy**

**Purpose**
Copy the current selection to the text clipboard.

**Declared In**
Field.h

**Prototype**
```c
void FldCopy (const FieldType *fldP)
```

**Parameters**
- `fldP` Pointer to a field object (FieldType structure).

**Result**
Returns nothing.

**Comments**
This function leaves the current selection highlighted.
This function replaces anything previously in the text clipboard if there is text to copy. If no text is selected, the function beeps and the clipboard remains intact.

See Also  FldCut, FldPaste

FldCut

Purpose  Copy the current selection to the text clipboard, delete the selection from the field, and redraw the field.

Declared In  Field.h

Prototype  void FldCut (FieldType *fldP)

Parameters  -> fldP  Pointer to a field object (FieldType structure).

Result  Returns nothing.

Comments  If text is selected, the text is removed from the field, the field’s dirty attribute is set, and anything previously in the text clipboard is replaced by the selected text.

If there is no selection or if the field is not editable, this function beeps.

See Also  FldCopy, FldPaste, FldUndo
**FldDelete**

**Purpose**
Delete the specified range of characters from the field and redraw the field.

**Declared In**
Field.h

**Prototype**
```c
void FldDelete (FieldType *fldP, UInt16 start, UInt16 end)
```

**Parameters**
- `-> fldP` Pointer to the field object (FieldType structure) to delete from.
- `-> start` The beginning of the range of characters to delete given as a valid byte offset into the field’s text string.
- `-> end` The end of the range of characters to delete given as a valid byte offset into the field’s text string. On systems that support multi-byte characters, this position must be an inter-character boundary. That is, it must not point to a middle byte of a multi-byte character.

**Result**
Returns nothing.

**Comments**
This function deletes all characters from the starting offset up to the ending offset and sets the field’s dirty attribute. It does not delete the character at the ending offset.

If `start` or `end` point to an intra-character boundary, FldDelete attempts to move the offset backward, toward the beginning of the text, until the offset points to an inter-character boundary (i.e., the start of a character).

FldDelete posts a [fldChangedEvent](#) to the event queue. If you call this function repeatedly, you may overflow the event queue with `fldChangedEvents`. An alternative is to remove the text...
handle from the field, change the text, and then set the field’s handle again. See FldGetTextHandle for a code example.

See Also  FldInsert, FldEraseField, TxtCharBounds

FldDirty

Purpose  Return true if the field has been modified since the text value was set.

Declared In  Field.h

Prototype  Boolean FldDirty (const FieldType *fldP)

Parameters  -> fldP  Pointer to a field object (FieldType structure).

Result  Returns true if the field has been modified either by the user or through calls to certain functions such as FldInsert and FldDelete, false if the field has not been modified.

See Also  FldSetDirty, FieldAttrType

FldDrawField

Purpose  Draw the text of the field.

Declared In  Field.h

Prototype  void FldDrawField (FieldType *fldP)

Parameters  -> fldP  Pointer to a field object (FieldType structure).

Result  Returns nothing.

Comments  The field’s usable attribute must be true or the field won’t be drawn.
            This function doesn’t erase the area behind the field before drawing.
If the field has the focus, the blinking insertion point is displayed in the field.

See Also  FldEraseField

**FldEraseField**

**Purpose**  Erase the text of a field and turn off the insertion point if it’s in the field.

**Declared In**  Field.h

**Prototype**  void FldEraseField (FieldType *fldP)

**Parameters**  -> fldP  Pointer to a field object (FieldType structure).

**Result**  Returns nothing.

**Comments**  You rarely need to call this function directly. Instead, use FrmHideObject, which calls FldEraseField for you.

This function visibly erases the field from the display, but it doesn’t modify the contents of the field or free the memory associated with it.

If the field has the focus, the blinking insertion point is turned off.

This function sets the visible attribute to false. (See FieldAttrType.)

See Also  FldDrawField
**FldFreeMemory**

**Purpose**
Release the handle-based memory allocated to the field’s text and the associated word-wrapping information.

**Declared In**
Field.h

**Prototype**
```c
void FldFreeMemory (FieldType *fldP)
```

**Parameters**
- `-> fldP` Pointer to a field object (`FieldType` structure).

**Result**
Returns nothing. May raise a fatal error message if the text associated with the field is actually a record in a database.

**Comments**
This function releases

- The memory allocated to the text of a field—the memory block that the `textHandle` member of the `FieldType` data structure points to.
  
  If the field’s `textHandle` is `NULL` but there is a text string associated with that field (which is often the case with noneditable text fields), the text string is not freed.

- The memory allocated to hold the word-wrapping information—the memory block that the `lines` member of the `FieldType` data structure points to.

This function doesn’t affect the display of the field. Fields allocate memory for the text string as needed, so it is not an error to call this function while the field is still displayed. That is, if `text` is `NULL` and the user starts typing in the field, the field simply allocates memory for text and continues.
**FldGetAttributes**

**Purpose**
Return the attributes of a field.

**Declared In**
Field.h

**Prototype**
void FldGetAttributes (const FieldType *fldP,
FieldAttrPtr attrP)

**Parameters**
- fldP Pointer to a FieldType structure.
- attrP Pointer to the FieldAttrType structure.

**Result**
Returns the field’s attributes in the attrP parameter.

**See Also**
FldSetAttributes

**FldGetBounds**

**Purpose**
Return the current bounds of a field.

**Declared In**
Field.h

**Prototype**
void FldGetBounds (const FieldType *fldP,
RectanglePtr rect)

**Parameters**
- fldP Pointer to a field object (FieldType structure).
- rect Pointer to a RectangleType structure.

**Result**
Returns nothing. Stores the field’s bounds in the RectangleType structure reference by rect.

**Comments**
Returns the rect field of the FieldType structure.

**See Also**
FldSetBounds, FrmGetObjectBounds
**FldGetFont**

**Purpose**
Return the ID of the font used to draw the text of a field.

**Declared In**
Field.h

**Prototype**
FontID FldGetFont (const FieldType *fldP)

**Parameters**
- `fldP` Pointer to a field object (`FieldType` structure).

**Result**
Returns the ID of the font.

**See Also**
FldSetFont

**FldGetInsPtPosition**

**Purpose**
Return the insertion point position within the string.

**Declared In**
Field.h

**Prototype**
UInt16 FldGetInsPtPosition (const FieldType *fldP)

**Parameters**
- `fldP` Pointer to a field object (`FieldType` structure).

**Result**
Returns the byte offset of the insertion point.

**Comments**
The insertion point is to the left of the byte offset that this function returns. That is, if this function returns 0, the insertion point is to the left of the first character in the string. In multiline fields, line feeds are counted as a single character in the string, and the byte offset after the line feed character is the beginning of the next line.

**See Also**
FldSetInsPtPosition
FldGetMaxChars

**Purpose**
Return the maximum number of bytes the field accepts.

**Declared In**
Field.h

**Prototype**
UInt16 FldGetMaxChars (const FieldType *fldP)

**Parameters**
-> fldP Pointer to a field object (FieldType structure).

**Result**
Returns the maximum length in bytes of characters the user is allowed to enter. This is the maxChars field in FieldType.

**See Also**
FldSetMaxChars

FldGetNumberOfBlankLines

**Purpose**
Return the number of blank lines that are displayed at the bottom of a field.

**Declared In**
Field.h

**Prototype**
UInt16 FldGetNumberOfBlankLines (const FieldType *fldP)

**Parameters**
-> fldP Pointer to a FieldType structure.

**Result**
Returns the number of blank lines visible.

**Comments**
This routine is useful for updating a scroll bar after characters have been removed from the text in a field. See the NoteViewScroll function in the Address sample application for an example.

**Compatibility**
Implemented only if 2.0 New Feature Set is present.
**FldGetScrollPosition**

**Purpose**
Return the offset of the first character in the first visible line of a field.

**Declared In**
Field.h

**Prototype**
```c
UInt16 FldGetScrollPosition (const FieldType *fldP)
```

**Parameters**
- `-> fldP` Pointer to a field object (`FieldType` structure).

**Result**
Returns the offset of the first visible character.

**See Also**
FldSetScrollPosition, LineInfoType

**FldGetScrollValues**

**Purpose**
Return the values necessary to update a scroll bar.

**Declared In**
Field.h

**Prototype**
```c
void FldGetScrollValues (const FieldType *fldP,
                        UInt16 *scrollPosP, UInt16 *textHeightP,
                        UInt16 *fieldHeightP)
```

**Parameters**
- `-> fldP` Pointer to a `FieldType` structure.
- `<- scrollPosP` The line of text that is the topmost visible line. Line numbering starts with 0.
- `<- textHeightP` The number of lines needed to display the field’s text, given the width of the field.
- `<- fieldHeightP` The number of visible lines in the field.

**Result**
Returns nothing. Stores the position, text height, and field height in the parameters passed in.
Comments

Use the values returned by this function to calculate the values you send to `SclSetScrollBar` to update the scroll bar. For example:

```c
FldGetScrollValues (fldP, &scrollPos,
    &textHeight, &fieldHeight);

if (textHeight > fieldHeight)
    maxValue = textHeight - fieldHeight;
else if (scrollPos)
    maxValue = scrollPos;
else
    maxValue = 0;

SclSetScrollBar (bar, scrollPos, 0, maxValue,
    fieldHeight-1);
```

Compatibility

Implemented only if 2.0 New Feature Set is present.

See Also

`FldSetScrollPosition`

**FldGetSelection**

**Purpose**

Return the current selection of a field.

**Declared In**

Field.h

**Prototype**

```c
void FldGetSelection (const FieldType *fldP,
    U16 *startPosition, U16 *endPosition)
```

**Parameters**

- `-> fldP` Pointer to a field object (FieldType structure).
- `<- startPosition` Pointer to the start of the selected characters range, given as the byte offset into the field’s text.
Fields
Field Functions

<- endPosition Pointer to end of the selected characters range given as the byte offset into the field’s text.

Result Returns the starting and ending byte offsets in startPosition and endPosition.

Comments The first character in a field is at offset zero.
If the user has selected the first five characters of a field, startPosition will contain the value 0 and endPosition the value 5, assuming all characters are a single byte long.

See Also FldSetSelection

FldGetTextAllocatedSize

Purpose Return the number of bytes allocated to hold the field’s text string. Don’t confuse this number with the actual length of the text string displayed in the field.

Declared In Field.h

Prototype UInt16 FldGetTextAllocatedSize (const FieldType *fldP)

Parameters -> fldP Pointer to a field object.

Result Returns the number of bytes allocated for the field’s text. This is the textBlockSize field in FieldType.

See Also FldSetTextAllocatedSize
FldGetTextHandle

Purpose
Return a handle to the block that contains the text string of a field.

Declared In
Field.h

Prototype
MemHandle FldGetTextHandle (const FieldType *fldP)

Parameters
-> fldP Pointer to a field object (FieldType structure).

Result
Returns the handle to the text string of a field or NULL if no handle has been allocated for the field pointer.

Comments
The handle returned by this function is not necessarily the handle to the start of the string. If you’ve used FldSetText to set the field’s text to a string that is part of a database record, the text handle points to the start of that record. You’ll need to compute the offset from the start of the record to the start of the string. You can either store the offset that you passed to FldSetText or you can compute the offset by performing pointer arithmetic on the pointer you get by locking this handle and the pointer returned by FldGetTextPtr.

If you are obtaining the text handle so that you can edit the field’s text, you must remove the handle from the field before you do so. If you change the text while it is being used by a field, the field’s internal structures specifying the text length, allocated size, and word wrapping information can become out of sync. To avoid this problem, remove the text handle from the field, change the text, and then set the field’s text handle again. For example:

    /* Get the handle for the string and unlock */
    /* it by removing it from the field. */
    textH = FldGetTextHandle(fldP);
    FldSetTextHandle (fldP, NULL);

    /* Insert code that modifies the string here. */
    /* The basic steps are: */
    /* resize the chunk if necessary, */
/* lock the chunk, write to it, and then */
/* unlock the chunk. If the text is in a */
/* database record, use Data Manager calls. */

/* Update the text in the field. */
FldSetTextHandle (fldP, textH);
FldDrawField(fldP);

See Also  FldSetTextHandle, FldGetTextPtr

FldGetTextHeight

Purpose  Return the height in pixels of the number of visible lines that are not empty.

Declared In  Field.h

Prototype  UInt16 FldGetTextHeight (const FieldType *fldP)

Parameters  -> fldP  Pointer to a field object (FieldType structure).

Result  Returns the height in pixels of the number of visible lines that are not empty.

Comments  Empty lines are all of the lines in the field following the last byte of text. Note that lines that contain only a linefeed are not empty. Also note that only lines that are visible are counted.

See Also  FldCalcFieldHeight
**FldGetTextLength**

**Purpose**
Return the length in bytes of the field’s text.

**Declared In**
Field.h

**Prototype**
`UInt16 FldGetTextLength (const FieldType *fldP)`

**Parameters**
- `fldP` Pointer to a field object (FieldType structure).

**Result**
Returns the length in bytes of a field’s text, not including the terminating null character. This is the `textLen` field of FieldType.

**FldGetTextPtr**

**Purpose**
Return a locked pointer to the field’s text string.

**Declared In**
Field.h

**Prototype**
`Char *FldGetTextPtr (const FieldType *fldP)`

**Parameters**
- `fldP` Pointer to a field object (FieldType structure).

**Result**
Returns a locked pointer to the field’s text string or NULL if the field is empty.

**Comments**
The pointer returned by this function can become invalid if the user edits the text after you obtain the pointer. Do not modify the contents of the pointer yourself. If you change the text while it is being used by a field, the field’s internal structures specifying the text length, allocated size, and word wrapping information can become out of sync. To avoid this problem, follow the instructions given under FldGetTextHandle.
**WARNING!** The pointer returned by this function is “owned” by the field until you specify a different pointer for the field. You should not store this pointer for future use, since the field can modify the size of the string, which can cause the pointer to become invalid.

**See Also**  
FldSetTextPtr, FldGetTextHandle

### FldGetVisibleLines

**Purpose**  
Return the number of lines that can be displayed within the visible bounds of the field, regardless of what text is stored in the field.

**Declared In**  
Field.h

**Prototype**  
UInt16 FldGetVisibleLines (const FieldType *fldP)

**Parameters**  
- fldP Pointer to a field object (FieldType structure).

**Result**  
Returns the number of lines the field displays. (This is the size of the lines array in the FieldType structure.)

**See Also**  
FldGetNumberOfBlankLines, FldCalcFieldHeight

### FldGrabFocus

**Purpose**  
Turn the insertion point on (if the specified field is visible) and position the blinking insertion point in the field.

**Declared In**  
Field.h

**Prototype**  
void FldGrabFocus (FieldType *fldP)

**Parameters**  
- fldP Pointer to a field object (FieldType structure).

**Result**  
Returns nothing.
Comments  
You rarely need to call this function directly. Instead, use \texttt{FrmSetFocus}, which calls \texttt{FldGrabFocus} for you.

One instance where you need to call \texttt{FldGrabFocus} directly is to programmatically set the focus in a field that is contained in a table cell.

This function sets the field attribute \texttt{hasFocus} to \texttt{true}. (See \texttt{FieldAttrType}.)

See Also  \texttt{FrmSetFocus, FldReleaseFocus}

\textbf{FldHandleEvent}

\textbf{Purpose}  
Handles events that affect the field, including the following: \texttt{keyDownEvent}, \texttt{penDownEvent}, and \texttt{fldEnterEvent}.

\textbf{Declared In}  \texttt{Field.h}

\textbf{Prototype}  
\texttt{Boolean FldHandleEvent (FieldType *fldP, EventType *eventP)}

\textbf{Parameters}  
\texttt{-&gt; fldP} Pointer to a field object (\texttt{FieldType} structure).
\texttt{-&gt; eventP} Pointer to an event (\texttt{EventType} data structure).

\textbf{Result}  
Returns \texttt{true} if the event was handled.

\textbf{Comments}  
When a \texttt{keyDownEvent} occurs in an editable text field, the keystroke appears in the field if it’s a printable character or manipulates the insertion point if it’s a “movement” character. The field is automatically updated.

When a \texttt{penDownEvent} occurs, the field sends a \texttt{f ldEnterEvent} to the event queue.

When a \texttt{fldEnterEvent} occurs, the field grabs the focus. If the user has tapped twice in the current location, the word at that location is selected. If the user has tapped three times, the entire line is selected. Otherwise, the insertion point is placed in the specified position.
When a `menuCmdBarOpenEvent` occurs, the field adds paste, copy, cut, and undo buttons to the command toolbar. These buttons are only added if they make sense in the current context. That is, the cut button is only added if the field is editable, the paste button is only added if there is text on the clipboard and the field is editable, and the undo button is only added if there is an action to undo.

If the event alters the contents of the field, this function visually updates the field.

This function doesn’t handle any events if the field is not editable or usable.

**Compatibility**

Double-tapping to select a word and triple-tapping to select a line are only supported if 3.5 New Feature Set is present.

`FldHandleEvent` only handles the `menuCmdBarOpenEvent` if 3.5 New Feature Set is present.

## FldInsert

**Purpose**

Replace the current selection if any with the specified string and redraw the field.

**Declared In**

Field.h

**Prototype**

```c
Boolean FldInsert (FieldType *fldP,
                   const Char *insertChars, UInt16 insertLen)
```

**Parameters**

- `-> fldP` Pointer to the field object (`FieldType` structure) to insert to.
- `-> insertChars` Text string to be inserted.
- `-> insertLen` Length in bytes of the text string to be inserted, not counting the trailing null character.

**Result**

Returns `true` if string was successfully inserted. Returns `false` if:

- The `insertLen` parameter is 0.
- The field is not editable.
Fields
Field Functions

- Adding the text would exceed the field’s size limit (the maxChars value).
- More memory must be allocated for the field, and the allocation fails.

Comments
If there is no current selection, the string passed is inserted at the position of the insertion point.
This function sets the field’s dirty attribute and posts a fldChangedEvent to the event queue. If you call this function repeatedly, you may overflow the event queue with fldChangedEvents. An alternative is to remove the text handle from the field, change the text, and then set the field’s handle again. See FldGetTextHandle for a code example.

See Also
FldPaste, FldDelete, FldCut, FldCopy

FldMakeFullyVisible

Purpose
Generates an event to cause a dynamically resizable field to expand its height to make its text fully visible.

Declared In
Field.h

Prototype
Boolean FldMakeFullyVisible (FieldType *fldP)

Parameters
-> fldP Pointer to a field object (FieldType structure).

Result
Returns true if the field is dynamically resizable and was not fully visible; false otherwise.

Comments
Use this function on a field whose dynamicSize attribute is true (see FieldAttrType).
This function does not actually resize the field. Instead, it computes how big the field should be to be fully visible and then posts this information to the event queue in a fldHeightChangedEvent.
**NOTE:** The event does not get generated if the number of lines in the field is equal to or greater than the value of the maximum lines attribute for the field.

If the field is contained in a table, the table’s code handles the `fldHeightChangedEvent`. If the field is directly on a form, your application code should handle the `fldHeightChangedEvent` itself. The form code does not handle the event for you. Note that the constant `maxFieldLines` defines the maximum number of lines a field can expand to if the field is using the standard font.

**See Also**  
TblHandleEvent

**FldNewField**

**Purpose**  
Create a new field object dynamically and install it in the specified form.

**Declared In**  
Field.h

**Prototype**  
`FieldType *FldNewField (void **formPP, UInt16 id, Coord x, Coord y, Coord width, Coord height, FontID font, UInt32 maxChars, Boolean editable, Boolean underlined, Boolean singleLine, Boolean dynamicSize, JustificationType justification, Boolean autoShift, Boolean hasScrollBar, Boolean numeric)`

**Parameters**  
`<- > formPP`  
Pointer to the pointer to the form in which the new field is installed. This value is not a handle; that is, the old form pointer value is not necessarily valid after this function returns. In subsequent calls, always use the new form pointer value returned by this function.
Fields
Field Functions

- **id** Symbolic ID of the field, specified by the developer. By convention, this ID should match the resource ID (not mandatory).

- **x** Horizontal coordinate of the upper-left corner of the field’s boundaries, relative to the window in which it appears.

- **y** Vertical coordinate of the upper-left corner of the field’s boundaries, relative to the window in which it appears.

- **width** Width of the field, expressed in pixels.

- **height** Height of the field, expressed in pixels.

- **font** Font to use to draw the field’s text.

- **maxChars** Maximum number of bytes held by the field this function creates.

- **editable** Pass `true` to create a field in which the user can edit text. Pass `false` to create a field that cannot be edited.

- **underlined** Pass `noUnderline` for no underline, or `grayUnderline` to have the field underline the text it displays. On Palm OS® version 3.1 and higher, pass `solidUnderline` to use a solid underline instead of a dotted underline.

- **singleLine** Pass `true` to create a field that can display only a single line of text.

- **dynamicSize** Pass `true` to create a field that resizes dynamically according to the amount of text it displays.

- **justification** Pass either of the values `leftAlign` or `rightAlign` to specify left justification or right justification, respectively. The `centerAlign` value is not supported.

- **autoShift** Pass `true` to specify the use of Palm OS 2.0 (and later) auto-shift rules.
-> hasScrollBar  Pass true to attach a scroll bar control to the field this function creates.

-> numeric  Pass true to specify that only characters in the range of 0 through 9 are allowed in the field.

**Result**  Returns a pointer to the new field object or NULL if there wasn’t enough memory to create the field. Out of memory situations could be caused by memory fragmentation.

**Compatibility**  Implemented only if 3.0 New Feature Set is present.

**See Also**  FrmValidatePtr, WinValidateHandle, CtlValidatePointer, FrmRemoveObject

### FldPaste

**Purpose**  Replace the current selection in the field, if any, with the contents of the text clipboard.

**Declared In**  Field.h

**Prototype**  void FldPaste (FieldType *fldP)

**Parameters**  

-> fldP  Pointer to a field object (FieldType structure).

**Result**  Returns nothing

**Comments**  The function performs these actions:

- Scrolls the field, if necessary, so the insertion point is visible.
- Inserts the clipboard text at the position of the insertion point if there is no current selection.
- Positions the insertion point after the last character inserted.
- Doesn’t delete the current selection if there is no text in the clipboard.

**See Also**  FldInsert, FldDelete, FldCut, FldCopy FldUndo
FldRecalculateField

Purpose  Update the structure that contains the word-wrapping information for each visible line.

Declared In  Field.h

Prototype  void FldRecalculateField (FieldType *fldP, Boolean redraw)

Parameters  
- > fldP  Pointer to a field object (FieldType structure).
- > redraw  If true, redraws the field.

Result  Returns nothing.

Comments  This function will allocate the memory block that contains the displayed lines information if, and only if, the block does not yet exist.

You should call this function when you change the field width or text length of the field. Do not call this function after changing the font or field height.

Note that many of the field functions, including FldSetTextHandle, FldInsert, and FldDelete, recalculate the word-wrapping information for you.

Compatibility  In releases prior to Palm OS 4.0, the word-wrapping information is only updated if the redraw parameter is set to true. As of Palm OS 4.0 it is updated whenever FldRecalculateField is called, regardless of the value of the redraw parameter.
**FldReleaseFocus**

**Purpose**  
Turn the blinking insertion point off if the field is visible and has the current focus, reset the Graffiti state, and reset the undo state.

**Declared In**  
Field.h

**Prototype**  
void FldReleaseFocus (FieldType *fldP)

**Parameters**  
- `-> fldP`  
  Pointer to a field object (FieldType structure).

**Result**  
Returns nothing.

**Comments**  
This function sets the field attribute hasFocus to false. (See FieldAttrType.)

Usually, you don’t need to call this function. If the field is in a form or in a table that doesn’t use custom drawing functions, the field code releases the focus for you when the focus changes to some other control. If your field is in any other type of object, such as a table that uses custom drawing functions or a gadget, you must call FldReleaseFocus when the focus moves away from the field.

**See Also**  
FldGrabFocus

**FldScrollable**

**Purpose**  
Return true if the field is scrollable in the specified direction.

**Declared In**  
Field.h

**Prototype**  
Boolean FldScrollable (const FieldType *fldP, WinDirectionType direction)

**Parameters**  
- `-> fldP`  
  Pointer to a field object (FieldType structure).
Fields
Field Functions

FldScrollField

Purpose
Scroll a field up or down by the number of lines specified.

Declared In
Field.h

Prototype
void FldScrollField (FieldType *fldP,
UInt16 linesToScroll, WinDirectionType direction)

Parameters
- > fldP Pointer to a field object (FieldType structure).
- > linesToScroll Number of lines to scroll.
- > direction The direction to scroll. DirectionType is
  defined in Window.h. It is an enum defining,
  among others, the constants winUp and winDown.

Result
Returns nothing.

Comments
This function can’t scroll horizontally, that is, right or left.

The field object is redrawn if it’s scrolled; however, the scrollbar is
not updated. Use SclSetScrollBar to update the scrollbar. For
example:

    FldScrollField (fldP, linesToScroll, direction);

    // Update the scroll bar.
    SclGetScrollBar (bar, &value, &min, &max,
Fields
Field Functions

&pageSize);

if (direction == winUp)
  value -= linesToScroll;
else
  value += linesToScroll;

SclSetScrollBar (bar, value, min, max,
                 pageSize);

If the field is not scrollable in the direction indicated, this function returns without performing any work. You can use FldScrollable before calling this function to see if the field can be scrolled.

See Also  FldScrollable, FldSetScrollPosition

FldSendChangeNotification

Purpose  Send a fldChangedEvent to the event queue.

Declared In  Field.h

Prototype  void FldSendChangeNotification
             (const FieldType *fldP)

Parameters  ->fldP  Pointer to a field object.

Result  Returns nothing.

Comments  This function is used internally by the field code. You normally never call it in application code.
**FldSendHeightChangeNotification**

**Purpose**
Send a `fldHeightChangedEvent` to the event queue.

**Declared In**
Field.h

**Prototype**
```c
void FldSendHeightChangeNotification (const FieldType *fldP, UInt16 pos, Int16 numLines)
```

**Parameters**
- `fldP` Pointer to a field object.
- `pos` Character position of the insertion point.
- `numLines` New number of lines in the field.

**Result**
Returns nothing.

**Comments**
This function is used internally by the field code. You normally never call it in application code.

**FldSetAttributes**

**Purpose**
Set the attributes of a field.

**Declared In**
Field.h

**Prototype**
```c
void FldSetAttributes (FieldType *fldP, const FieldAttrType *attrP)
```

**Parameters**
- `fldP` Pointer to a `FieldType` structure.
- `attrP` Pointer to the attributes.

**Result**
Returns nothing.

**Comments**
This function does not do anything to make the new attribute values take effect. For example, if you use this function to change the value of the underline attribute, you won’t see its effect until you call `FldDrawField`.
You usually do not have to modify field attributes at runtime, so you rarely need to call this function.

**WARNING!** You must not call this function to change any attributes that are noted as “for internal use only.”

The proper way to use `FldSetAttributes` is to:
1. Call `FldGetAttributes` to retrieve the attributes.
2. Set the specific flags that you want to modify.
3. Call `FldSetAttributes` to make the modifications.

**WARNING!** You must not call any field routines between calling `FldGetAttributes` and `FldSetAttributes`; this can cause the attributes to be out of sync, with unpredictable results.

**See Also**  
`FldGetAttributes`, `FieldAttrType`

### FldSetBounds

**Purpose**  
Change the position or size of a field.

**Declared In**  
Field.h

**Prototype**  
```c
void FldSetBounds (FieldType *fldP,  
const RectangleType *rP)
```

**Parameters**
- `-> fldP`  
  Pointer to a field object (`FieldType` structure).
- `-> rP`  
  Pointer to a `RectangleType` structure that contains the new bounds of the display.

**Result**  
Returns nothing. May raise a fatal error message if the memory block that contains the word-wrapping information needs to be resized and there is not enough space to do so.

**Comments**  
If the field is visible, the field is redrawn within its new bounds.
NOTE: You can change the height or location of the field while it's visible, but do not change the width.
The memory block that contains the word-wrapping information (see LineInfoType) will be resized if the number of visible lines is changed. The insertion point is assumed to be off when this routine is called.

Make sure that rect is at least as tall as a single line in the current font. (You can determine this value by calling FntLineHeight.) If it’s not, results are unpredictable.

See Also FldGetBounds, FrmSetObjectBounds

FldSetDirty

Purpose Set whether the field has been modified.

Declared In Field.h

Prototype void FldSetDirty (FieldType *fldP, Boolean dirty)

Parameters -> fldP Pointer to a field object (FieldType structure).
-> dirty true if the text is modified.

Result Returns nothing.

Comments You typically call this function when you want to clear the dirty attribute. The dirty attribute is set when the user enters or deletes text in the field. It is also set by certain field functions, such as FldInsert and FldDelete.

See Also FldDirty
**FldSetFont**

**Purpose**
Set the font used by the field, update the word-wrapping information, and draw the field if the field is visible.

**Declared In**
Field.h

**Prototype**
void FldSetFont (FieldType *fldP, FontID fontID)

**Parameters**
- `fldP` Pointer to a field object (**FieldType** structure).
- `fontID` ID of new font.

**Result**
Returns nothing.

**See Also**
FldGetFont, FieldAttrType

**FldSetInsertionPoint**

**Purpose**
Set the location of the insertion point based on a specified string position.

**Declared In**
Field.h

**Prototype**
void FldSetInsertionPoint (FieldType *fldP, Uint16 pos)

**Parameters**
- `fldP` Pointer to a **FieldType** structure.
- `pos` New location of the insertion point, given as a valid offset in bytes into the field’s text. On systems that support multi-byte characters, you must make sure that this specifies an inter-character boundary (does not specify the middle or end bytes of a multi-byte character).

**Result**
Nothing.
Comments
This routine differs from `FldSetInsPtPosition` in that it doesn’t make the character position visible. `FldSetInsertionPoint` also doesn’t make the field the current focus of input if it was not already.

If `pos` indicates a position beyond the end of the text in the field, the insertion point is set to the end of the field’s text.

Compatibility
Implemented only if [2.0 New Feature Set](#) is present.

See Also
`TxtCharBounds`

### FldSetInsPtPosition

**Purpose**
Set the location of the insertion point for a given string position.

**Declared In**
Field.h

**Prototype**
```c
void FldSetInsPtPosition (FieldType *fldP, UInt16 pos)
```

**Parameters**
- `-> fldP`
  Pointer to a field object (FieldType structure).
- `-> pos`
  New location of the insertion point, given as a valid offset in bytes into the field’s text. On systems that support multi-byte characters, you must make sure that this specifies an inter-character boundary (does not specify the middle or end bytes of a multi-byte character).

**Result**
Returns nothing.

**Comments**
If the position is beyond the visible text, the field is scrolled until the position is visible.

**See Also**
`FldGetInsPtPosition`, `TxtCharBounds`
**FldSetMaxChars**

**Purpose**
Set the maximum number of bytes the field accepts (the `maxChars` value).

**Declared In**
Field.h

**Prototype**
```c
void FldSetMaxChars (FieldType *fldP, UInt16 maxChars)
```

**Parameters**
- `fldP` Pointer to a field object (`FieldType` structure).
- `maxChars` Maximum size in bytes of the characters the user may enter. You may specify any value up to `maxFieldTextLen`.

**Result**
Returns nothing.

**Comments**
Line feed characters are counted when the length of characters is determined.

**See Also**
FldGetMaxChars

---

**FldSetMaxVisibleLines**

**Purpose**
Allows the creation of tables and fields smaller than 121 pixels tall that still drag-select when there are more lines of text than will fit in the space provided.

**Declared In**
Field.h

**Prototype**
```c
void FldSetMaxVisibleLines (FieldType *fldP, Uint8 maxLines)
```

**Parameters**
- `fldP` Pointer to a field object (`FieldType` structure).
-> maxLines Maximum number of lines to which the field will visually grow.

Result Returns nothing.

Comments A field can be dynamically expandable. When it is, the field package needs to know the maximum number of lines that should be visible so it can prevent the field from being expanded further. Since field expansion is actually handled by enclosing objects—tables or forms—this function’s primary purpose is to allow the enclosing object to tell the field how big it can get.

By default, tables assume that the field can get as big as the table. If you don’t call this function, fields expect to be at least 121 pixels tall and try to grow repeatedly until they are.

FldSetScrollPosition

Purpose Scroll the field such that the character at the indicated offset is the first character on the first visible line. Redraw the field if necessary.

Declared In Field.h

Prototype void FldSetScrollPosition (FieldType *fldP, UInt16 pos)

Parameters -> fldP Pointer to a field object (FieldType structure).
-> pos Byte offset into the field’s text string of first character to be made visible. On systems that support multi-byte characters, you must make sure that this specifies an inter-character boundary (does not specify the middle or end bytes of a multi-byte character).

Result Returns nothing.

Comments This function scrolls the field but does not update the field’s scrollbar. You should update the scrollbar after calling this function.
To do so, first call `FldGetScrollValues` to determine the values to use, and then call `SclSetScrollBar`.

**See Also**  
`FldGetScrollPosition, FldScrollField, TxtCharBounds`

## FldSetSelection

**Purpose**  
Set the current selection in a field and highlight the selection if the field is visible.

**Declared In**  
Field.h

**Prototype**  
```c
void FldSetSelection (FieldType *fldP,  
UInt16 startPosition, UInt16 endPosition)
```

**Parameters**

- `-> fldP`  
  Pointer to a field object (`FieldType` structure).

- `-> startPosition`  
  Starting offset of the character range to highlight, given as a byte offset into the field’s text.

- `-> endPosition`  
  Ending offset of the character range to highlight. The ending offset should be greater than or equal to the starting offset. On systems that support multi-byte characters, this position must be an inter-character boundary. That is, it must not point to a middle byte of a multi-byte character.

**Result**  
Returns nothing.

**Comments**

To cancel a selection, set both `startPosition` and `endPosition` to the same value. If `startPosition` equals `endPosition`, then the current selection is unhighlighted.

If either `startPosition` or `endPosition` point to an intra-character boundary, `FldSetSelection` attempts to move that
offset backward, toward the beginning of the string, until the offset points to an inter-character boundary (i.e., the start of a character).

See Also  
TxtCharBounds

**FldSetText**

**Purpose** Set the text value of the field without updating the display.

**Declared In** Field.h

**Prototype**

```c
void FldSetText (FieldType *fldP,
MemHandle textHandle, UInt16 offset, UInt16 size)
```

**Parameters**

- `-> fldP` Pointer to a field object (`FieldType` structure).
- `-> textHandle` Unlocked handle of a block containing a null-terminated text string. Pass NULL for this parameter to remove the association between the field and the string it is currently displaying so that the string is not freed with the field when the form is deleted.
- `-> offset` Offset from start of block to start of the text string.
- `-> size` The allocated size of the text string. This is not the string length, and should not be set to 0, unless you are setting the text to the empty string.

**Result** Returns nothing.

**Comments**

This function allows applications to perform editing in place in memory. You can use it to point the field to a string in a database record so that you can edit that string directly using field routines. As characters are added to the field’s text, the block that contains the text is grown. So that the block doesn’t have to be expanded for each character, it is expanded several bytes at a time; this expansion may result in some unused space in the text block. As characters are
removed from the field’s text, the space is not automatically
reclaimed. Because adding or removing characters when editing a
data record in place may result in unused space at the end of the
field’s text block, applications should call \texttt{FldCompactText}
on before the field is unlocked to release any unused space.

The handle that you pass to this function is assumed to contain a
null-terminated string starting at \texttt{offset} bytes in the memory
chunk. The string should be between 0 and \texttt{size} - 1 bytes in length.
The field does not make a copy of the memory chunk or the string
data; instead, it stores the handle to the record in its structure.

\begin{warn}
\textbf{WARNING!} You cannot use this function to set two fields on a
form so that they share a single string value. Thus, for instance, if
you have a single string containing a person’s name you cannot
call \texttt{FldSetText} twice with the same string (but a different
offset) to set up a first name field and a last name field.
\end{warn}

\texttt{FldSetText} updates the word-wrapping information and places
the insertion point after the last visible character, but it does not
update the display. You must call \texttt{FldDrawField} after calling this
function to update the display.

\texttt{FldSetText} increments the lock count for \texttt{textHandle} and
decrements the lock count of its previous text handle (if any).

Because \texttt{FldSetText} (and \texttt{FldSetTextHandle}) may be used to
edit database records, they do not free the memory associated with
the previous text handle. If the previous text handle points to a
string on the dynamic heap and you want to free it, use
\texttt{FldGetTextHandle} to obtain the handle before using
\texttt{FldSetText} and then free that handle after using \texttt{FldSetText}.
(See \texttt{FldGetTextHandle} for a code example.)

If the field points to a database record, you want the memory
associated with the text handle to persist; however, this memory
and all other memory associated with the field is freed when the
field itself is freed, which happens when the form is closed. If you
don’t want the memory associated with the text handle freed when
the field is freed, use \texttt{FldSetText} and pass \texttt{NULL} for the text
handle immediately before the form is closed. Passing \texttt{NULL}
removes the association between the field and the text handle that you want retained. That text handle is unlocked as a result of the FldSetText call, and when the field is freed, there is no text handle to free with it.

See Also  FldSetTextPtr, FldSetTextHandle

FldSetTextAllocatedSize

Purpose  Set the number of bytes allocated to hold the field’s text string. Don’t confuse this with the actual length of the text string displayed in the field.

Declared In  Field.h

Prototype  void FldSetTextAllocatedSize (FieldType *fldP, UInt16 allocatedSize)

Parameters  -> fldP  Pointer to a field object (FieldType structure).
-> allocatedSize  Number of bytes to allocate for the text.

Result  Returns nothing.

Comments  This function generally is not used. It does not resize the field’s allocated memory for the text string; it merely sets the textBlockSize field of the FieldType structure. The value of this field is computed and maintained internally by the field, so you should not have to call FldSetTextAllocatedSize directly.

See Also  FldGetTextAllocatedSize, FldCompactText
**FldSetTextHandle**

**Purpose**
Set the text value of a field to the string associated with the specified handle. Does not update the display.

**Declared In**
Field.h

**Prototype**
```c
void FldSetTextHandle (FieldType *fldP, MemHandle textHandle)
```

**Parameters**
- `-> fldP` Pointer to a field object (**FieldType** structure).
- `-> textHandle` Unlocked handle of a field’s text string. Pass NULL for this parameter to remove the association between the field and the string it is currently displaying so that the string is not freed with the field when the form is deleted.

**Result**
Returns nothing.

**Comments**
This function differs from **FldSetText** in that it uses the entire memory chunk pointed to by `textHandle` for the string. In fact, this function simply calls **FldSetText** with an offset of 0 and a size equal to the entire length of the memory chunk. Use it to have the field edit a string in a database record if the entire record consists of that string, or use it to have the field edit a string in the dynamic heap.

As characters are added to the field’s text, the block that contains the text is grown. So that the block doesn’t have to be expanded for each character, it is expanded several bytes at a time; this expansion may result in some unused space in the text block. As characters are removed from the field’s text, the space is not automatically reclaimed. Because adding or removing characters when editing a data record in place may result in unused space at the end of the field’s text block, applications should call **FldCompactText** on before the field is unlocked to release any unused space.

**FldSetTextHandle** updates the word-wrapping information and places the insertion point after the last visible character, but it does...
not update the display. You must call `FldDrawField` after calling this function to update the display.

`FldSetTextHandle` increments the lock count for `textHandle` and decrements the lock count of its previous text handle (if any).

Because `FldSetTextHandle` (and `FldSetText`) may be used to edit database records, they do not free the memory associated with the previous text handle. If the previous text handle points to a string on the dynamic heap and you want to free it, use `FldGetTextHandle` to obtain the handle before using `FldSetText` and then free that handle after using `FldSetText`. For example:

```c
/* get the old text handle */
oldTxtH = FldGetTextHandle(fldP);

/* change the text and update the display */
FldSetTextHandle(fldP, txtH);
FldDrawField(fldP);

/* free the old text handle */
if (oldTxtH != NULL)
    MemHandleFree(oldTxtH);
```

If the field points to a database record, you want the memory associated with the text handle to persist; however, this memory and all other memory associated with the field is freed when the field itself is freed, which happens when the form is closed. If you don’t want the memory associated with the text handle freed when the field is freed, use `FldSetTextHandle` and pass `NULL` for the text handle immediately before the form is closed. Passing `NULL` removes the association between the field and the text handle that you want retained. That text handle is unlocked as a result of the `FldSetTextHandle` call, and when the field is freed, there is no text handle to free with it.

See Also  
`FldSetTextPtr`, `FldSetText`
**FldSetTextPtr**

**Purpose**  Set a noneditable field’s text to point to the specified text string.

**Declared In**  Field.h

**Prototype**  
void FldSetTextPtr (FieldType *fldP, Char *textP)

**Parameters**  
- `fldP`  Pointer to a field object (FieldType structure).
- `textP`  Pointer to a null-terminated string.

**Result**  Returns nothing. May display an error message if passed an editable text field.

**Comments**  
Do not call FldSetTextPtr with an editable text field. Instead, call FldSetTextHandle for editable text fields. FldSetTextPtr is intended for displaying noneditable text in the user interface.

If the field has more than one line, use FldRecalculateField to recalculate the word wrapping.

This function does not visually update the field. Use FldDrawField to do so.

The field never frees the string that you pass to this function, even when the field itself is freed. You must free the string yourself. Before you free the string, make sure the field is not still displaying it. Set the field’s string pointer to some other string or call FldSetTextPtr(fldP, NULL) before freeing a string you have passed using this function.

**See Also**  FldSetTextHandle, FldGetTextPtr
### FldSetUsable

**Purpose**
Set a field to usable or nonusable.

**Declared In**
Field.h

**Prototype**
```c
void FldSetUsable (FieldType *fldP,
                   Boolean usable)
```

**Parameters**
- `fldP` Pointer to a `FieldType` structure.
- `usable` true to set usable; false to set nonusable.

**Result**
Returns nothing.

**Comments**
A nonusable field doesn’t display or accept input. Use `FrmHideObject` and `FrmShowObject` instead of using this function.

**See Also**
FldEraseField, FldDrawField, FieldAttrType

### FldUndo

**Purpose**
Undo the last change made to the field object, if any. Changes include typing, backspaces, delete, paste, and cut.

**Declared In**
Field.h

**Prototype**
```c
void FldUndo (FieldType *fldP)
```

**Parameters**
- `fldP` Pointer to the field (`FieldType` structure) that has the focus.

**Result**
Returns nothing.

**See Also**
FldPaste, FldCut, FldDelete, FldInsert
**FldWordWrap**

**Purpose**  Given a string and a width, return the number of bytes of characters that can be displayed using the current font.

**Declared In**  Field.h

**Prototype**  
```c
UInt16 FldWordWrap (const Char *chars, Int16 maxWidth)
```

**Parameters**
- `chars`  Pointer to a null-terminated string.
- `maxWidth`  Maximum line width in pixels.

**Result**  Returns the length in bytes of the characters that can be displayed.

**See Also**  FntWordWrap
Fields
Field Functions
This chapter describes the global find facility API declared in the header file Find.h.

Find Functions

**FindDrawHeader**

**Purpose**
Draw the header line that separates, by application, the list of found items.

**Declared In**
Find.h

**Prototype**
Boolean FindDrawHeader (FindParamsPtr findParams, Char const* title)

**Parameters**
- `findParams` Pointer to the `sysAppLaunchCmdFind` launch code’s parameter block.
- `title` String to display as the title for the current application.

**Result**
Returns true if Find screen is filled up. Applications should exit from the search if this occurs.

**Comments**
Call this function once at the beginning of your application’s response to the `sysAppLaunchCmdFind` launch code. This function draws a header for your application’s Find results. The header separates the search results from your application with the search results from another application.

If your application searches multiple databases, you may also use `FindDrawHeader` as a separator between databases.
Find
Find Functions

FindGetLineBounds

Purpose
Returns the bounds of the next available line for displaying a match in the Find results dialog.

Declared In
Find.h

Prototype
void FindGetLineBounds
    (const FindParamsType *findParams, RectanglePtr r)

Parameters
-> findParams  Pointer to the sysAppLaunchCmdFind launch code’s parameter block.
<- r  The bounds of the area that should contain the next line of results.

Result
Returns nothing.

FindSaveMatch

Purpose
Saves the record and position within the record of a text search match. This information is saved so that it’s possible to later navigate to the match.

Declared In
Find.h

Prototype
Boolean FindSaveMatch (FindParamsPtr findParams,
    UInt16 recordNum, UInt16 pos, UInt16 fieldNum,
    UInt32 appCustom, UInt16 cardNo, LocalID dbID)

Parameters
-> findParams  Pointer to the sysAppLaunchCmdFind launch code’s parameter block.
-> recordNum  Record index. This parameter sets the recordNum field in the sysAppLaunchCmdGoto’s parameter block.
-> pos  Offset of the match string from start of record. This parameter sets the matchPos field in the sysAppLaunchCmdGoto’s parameter block.
Find

Find Functions

- fieldNum
  Field number that the string was found in. This parameter sets the matchFieldNum field in the sysAppLaunchCmdGoto’s parameter block.

- appCustom
  Extra data the application can save with a match. This parameter sets the matchCustom field in the sysAppLaunchCmdGoto’s parameter block.

- cardNo
  Card number of the database that contains the match. This parameter sets the dbCardNo field in the sysAppLaunchCmdGoto’s parameter block.

- dbID
  Local ID of the database that contains the match. This parameter sets the dbID field in the sysAppLaunchCmdGoto’s parameter block.

Result
Returns true if Find screen is filled up. Applications should exit from the search if this occurs.

Comments
Call this function when your application finds a record with a matching string (FindStrInStr or TxtFindString returns true). This function saves the information you pass. If the user clicks this selection in the Find results dialog, the information is retrieved and used to set up the sysAppLaunchCmdGoto launch code’s parameter block.

You can use the appCustom field for any application-specific data that might be needed to navigate to the record if the user selects it. It’s common for localizable applications to set appCustom to the length of the matching string because the global find facility cannot correctly determine the length of the matching string on systems with multi-byte character sets. In some character encodings, one character may be accurately represented as either a single-byte character or a multi-byte character. The TxtFindString function accurately matches single-byte characters against their multi-byte equivalents and returns the length of the matching string. If you passTxtFindString’s return value as the appCustom parameter to FindSaveMatch, the matchCustom field of the
sysAppLaunchCmdGoTo parameter block contains the length of the matching string.

If your application requires more custom information, you can store the information in a feature and store the feature number in the appCustom field. See the “Feature Manager” chapter for more information.

**FindStrInStr**

**Purpose** Perform a case-blind prefix search for a string in another string. This function assumes that the string to find has already been normalized for searching.

**Declared In** Find.h

**Prototype**

```c
Boolean FindStrInStr (Char const *strToSearch, Char const *strToFind, UInt16 *posP)
```

**Parameters**

- **-> strToSearch** String to search.
- **-> strToFind** Normalized version of the text string to be found.
- **<- posP** If a match is found, contains the offset of the match within strToSearch.

**Result**

Returns true if the string was found. FindStrInStr matches the beginnings of words only; that is, strToFind must be a prefix of one of the words in strToSearch for FindStrInStr to return true.

**Comment**

Don’t use this function on systems that support the text manager. Instead, use **TxtFindString**, which performs searches on strings that contain multi-byte characters and returns the length of the matching text.

For backward compatibility with systems that don’t support the text manager, use **TxtGlueFindString**, found in the PalmOSGlue library. TxtGlueFindString calls TxtFindString if the text
manager is present, or `FindStrInStr` if it is not present. For more information, see Chapter 75, "PalmOSGlue Library."

The method by which a search string is normalized varies depending on the version of Palm OS® and the character encoding supported by the device. The string passed to your application in the `strToFind` field of the `sysAppLaunchCmdFind` launch code parameter block has already been normalized. It can be passed directly to `FindStrInStr`, `TxtFindString`, or `TxtGlueFindString`. If you need to create your own normalized search string, use `TxtGluePrepFindString`, also in the PalmOSGlue library.
Forms

This chapter provides the following information about form objects:

- Form Data Structures
- Form Constants
- Form Resources
- Form Functions
- Application-Defined Functions

The header file `Form.h` declares the API that this chapter describes. For more information on forms, see the section “Text” in the Palm OS Programmer’s Companion, vol. I.

Form Data Structures

FormAttrType

The FormAttrType bit field defines the visible characteristics of the form.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the FormAttrType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```c
typedef struct {
    Uint16 usable :1;
    Uint16 enabled :1;
    Uint16 visible :1;
    Uint16 dirty :1;
    Uint16 saveBehind :1;
    Uint16 graffitiShift :1;
} FormAttrType;
```
Forms
Form Data Structures

```c
UInt16 globalsAvailable : 1;
UInt16 doingDialog : 1;
UInt16 exitDialog : 1;
UInt16 reserved : 7;
UInt16 reserved2;
} FormAttrType;
```

Your code should treat the `FormAttrType` bit field as opaque. Do not attempt to change bit field member values directly.

Field Descriptions

- **usable**: Not set if the form is not considered part of the current interface of the application, and it doesn’t appear on screen.
- **enabled**: Not used.
- **visible**: Set or cleared internally when the field object is drawn or erased.
- **dirty**: Not used.
- **saveBehind**: Set if the bits behind the form are saved when the form is drawn.
- **graffitiShift**: Set if the graffiti shift indicator is supported.
- **globalsAvailable**: System use only.
- **doingDialog**: System use only.
- **exitDialog**: System use only.
- **reserved**: Reserved for system use.
- **reserved2**: Reserved for system use.

Compatibility

The `globalsAvailable`, `doingDialog`, and `exitDialog` flags are present only if 3.5 New Feature Set is present.
FormBitmapType

The FormBitmapType structure defines the visible characteristics of a bitmap on a form.

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the FormBitmapType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```c
typedef struct {
    FormObjAttrType  attr;
    PointType        pos;
    UInt16           rscID;
} FormBitmapType;
```

**Field Descriptions**

- **attr** See FormObjAttrType.
- **pos** Location of the bitmap.
- **rscID** Resource ID of the bitmap. If you use DmGetResource with this value as the resource ID, it returns a pointer to a BitmapType structure.

FormFrameType

The FormFrameType structure defines a frame that appears on the form.

```c
typedef struct {
    UInt16           id;
    FormObjAttrType  attr;
    RectangleType    rect;
    UInt16           frameType;
} FormFrameType;
```
Field Descriptions

id  ID of the frame.
attr  See FormObjAttrType.
rect  Location and size of the frame.
frameType  The type of frame.

FormGadgetAttrType
The FormGadgetAttrType bit field defines a gadget’s attributes.

WARNING!  PalmSource, Inc. does not support or provide backward compatibility for the FormGadgetAttrType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

typedef struct {
   UInt16 usable : 1;
   UInt16 extended : 1;
   UInt16 visible : 1;
   UInt16 reserved : 13;
} FormGadgetAttrType;

Your code should treat the FormGadgetAttrType structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.
Field Descriptions

usable  Not set if the gadget is not considered part of the current interface of the application, and it doesn’t appear on screen. This is set by `FrmShowObject` and cleared by `FrmHideObject`.

extended  If set, the gadget is an extended gadget. Extended gadgets are supported if 3.5 New Feature Set is present. An extended gadget has the handler field defined in its `FormGadgetType`. If not set, the gadgets is a standard gadget compatible with all releases of Palm OS®.

visible  Set or cleared when the gadget is drawn or erased. `FrmHideObject` clears this value. You should set it explicitly in the gadget’s callback function (if it has one) in response to a draw request.

reserved  Reserved for future use.

Many form functions (`FrmGetObjectType`, `FrmHideObject`, and `FrmGetObjectBounds`, for example) take an object index as one of their arguments. The most common way to get an object’s index is to call `FrmGetObjectIndex`. `FrmGetObjectIndex` takes a form ID and returns the form object’s index. This is the routine one should use in most cases, because the application usually knows the object ID. However, gadgets and specifically extended gadgets, have APIs with callbacks that pass back the gadget pointer and not the ID. In those cases, the only way to get the object index (so one can use the `FrmGetObject*` APIs) is to use the function `FrmGetObjectIndexFromPtr`.

If you need the same functionality on pre-Palm OS 4.0 systems then you can accomplish the same thing with the following code snippet.

```c
UInt16 index;
UInt16 objIndex = frmInvalidObjectId;
UInt16 numObjects = FrmGetNumberOfObjects(frmP)
for (index = 0; index < numObjects; index++) {
    if (FrmGetObjectPtr(index) == myObjPtr) {
        // Found it
        objIndex = index;
        break;
    }
```
Forms
Form Data Structures

} } Compatibility

This type is defined only if 3.5 New Feature Set is present.
FormGadgetType

The FormGadgetType structure defines a gadget object that appears on a form.

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the FormGadgetType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```c
typedef struct {
    UInt16 id;
    FormGadgetAttrType attr;
    RectangleType rect;
    const void *data;
    FormGadgetHandlerType *handler;
} FormGadgetType;
```

Your code should treat the FormGadgetType structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

**Field Descriptions**

- **id**: ID of the gadget resource.
- **attr**: See `FormGadgetAttrType`.
- **rect**: Location and size of the object.
- **data**: Pointer to any specific data that needs to be stored. You can set and retrieve the value of this field with `FrmGetGadgetData` and `FrmSetGadgetData`.
- **handler**: Pointer to a callback function that controls the gadget’s behavior and responds to events. You can set this field with `FrmSetGadgetHandler`.

Many form functions (`FrmGetObjectType`, `FrmHideObject`, and `FrmGetObjectBounds`, for example) take an object index as one of their arguments. The most common way to get an object’s index is to
call \texttt{FrmGetObjectIndex}. \texttt{FrmGetObjectIndex} takes a form ID and returns the form object's index. This is the routine one should use in most cases, because the application usually knows the object ID. However, gadgets have APIs with callbacks that pass back the gadget pointer and not the ID. In those cases, the only way to get the object index (so one can use the \texttt{FrmGetObject*} APIs) is to use the function \texttt{FrmGetObjectIndexFromPtr}.

If you need the same functionality on pre-Palm OS 4.0 systems then you can accomplish the same thing with the following code snippet.

```c
UInt16 index;
UInt16 objIndex = frmInvalidObjectId;
UInt16 numObjects = FrmGetNumberOfObjects(frmP)
for (index = 0; index < numObjects; index++) {
    if (FrmGetObjectPtr(index) == myObjPtr) {
        // Found it
        objIndex = index;
        break;
    }
}
```

\textbf{Compatibility} In Palm OS\textsuperscript{®} releases prior to 3.5, the \texttt{attr} field was of type \texttt{FormObjAttrType} and the \texttt{handler} field did not exist.

\textbf{FormGadgetTypeInCallback}

The \texttt{FormGadgetTypeInCallback} structure is passed to your extended gadget handler and is identical to \texttt{FormGadgetType} except that its contents are not hidden when \texttt{DO_NOT_ALLOW_ACCESS_TO_INTERNALS_OF_STRUCTS} is defined. This allows you to freely access the contents of an extended gadget structure from within your extended gadget callback functions.

```c
typedef struct {
    UInt16 id;
    FormGadgetAttrType attr;
    RectangleType rect;
    const void *data;
    FormGadgetHandlerType *handler;
} FormGadgetTypeInCallback;
```
Field Descriptions

id  ID of the gadget resource.
attr See FormGadgetAttrType.
rect Location and size of the object.
data Pointer to any specific data that needs to be stored.
handler Pointer to a callback function that controls the gadget’s behavior and responds to events.

Many form functions (FrmGetObjectIndex, FrmHideObject, and FrmGetObjectBounds, for example) take an object index as one of their arguments. The most common way to get an object’s index is to call FrmGetObjectIndex. FrmGetObjectIndex takes a form ID and returns the form object’s index. This is the routine one should use in most cases, because the application usually knows the object ID. However, extended gadgets have APIs with callbacks that pass back the gadget pointer and not the ID. In those cases, the only way to get the object index (so one can use the FrmGetObject* APIs) is to use the function FrmGetObjectIndexFromPtr.

If you need the same functionality on pre-Palm OS 4.0 systems then you can accomplish the same thing with the following code snippet.

```c
UInt16 index;
UInt16 objIndex = frmInvalidObjectId;
UInt16 numObjects = FrmGetNumberOfObjects(frmP)
for (index = 0; index < numObjects; index++) {
    if (FrmGetObjectPtr(index) == myObjPtr) {
        // Found it
        objIndex = index;
        break;
    }
}
```

Compatibility  Introduced in the Palm OS 4.0 SDK Update 1.

FormLabelType

The FormLabelType structure defines a label that appears on a form.
**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `FormLabelType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```c
typedef struct {
    UInt16           id;
    PointType        pos;
    FormObjAttrType  attr;
    FontID           fontID;
    UInt8            reserved;
    Char             *text;
} FormLabelType;
```

Your code should treat the `FormLabelType` structure as opaque. Do not attempt to change structure member values directly.

**Field Descriptions**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Resource ID of the label.</td>
</tr>
<tr>
<td>pos</td>
<td>Location of the label.</td>
</tr>
<tr>
<td>attr</td>
<td>See <code>FormObjAttrType</code>.</td>
</tr>
<tr>
<td>fontID</td>
<td>Font ID of the font used for the label.</td>
</tr>
<tr>
<td>reserved</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>text</td>
<td>Text of the label.</td>
</tr>
</tbody>
</table>

**FormLineType**

The `FormLineType` structure defines a line appearing on a form.

```c
typedef struct {
    FormObjAttrType  attr;
    PointType        point1;
    PointType        point2;
} FormLineType;
```
Your code should treat the `FormLineType` structure as opaque. Do not attempt to change structure member values directly.

**Field Descriptions**

- **attr**
  See [FormObjAttrType](#).
- **point1**
  Starting point of the line.
- **point2**
  Ending point of the line.

### FormObjAttrType

The `FormObjAttrType` bit field defines a form object’s attributes.

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `FormObjAttrType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```c
typedef struct {
    UInt16 usable : 1;
    UInt16 reserved : 15;
} FormObjAttrType;
```

Your code should treat the `FormObjAttrType` structure as opaque. Do not attempt to change structure member values directly.

**Field Descriptions**

- **usable**
  Not set if the object is not considered part of the current interface of the application, and it doesn’t appear on screen.
- **reserved**
  Reserved for future use.

### FormObjectKind

The `FormObjectKind` enum specifies values for the `objectType` field of the `FormObjListType`. It specifies how to interpret the object field.
enum formObjects {
    frmFieldObj,
    frmControlObj,
    frmListObj,
    frmTableObj,
    frmBitmapObj,
    frmLineObj,
    frmFrameObj,
    frmRectangleObj,
    frmLabelObj,
    frmTitleObj,
    frmPopupObj,
    frmGraffitiStateObj,
    frmGadgetObj,
    frmScrollbarObj,
};

typedef enum formObjects FormObjectKind;

Value Descriptions

frmFieldObj  Text field
frmControlObj Control
frmListObj   List
frmTableObj  Table
frmBitmapObj Form bitmap
frmLineObj   Line
frmFrameObj  Frame
frmRectangleObj Rectangle
frmLabelObj  Label
frmTitleObj  Form title
frmPopupObj  Popup list
frmGraffitiStateObj Graffiti® state indicator
### Form ObjectType

The `FormObjectType` union points to the C structure for a user interface object that appears on the form.

```c
typedef union {
    void *ptr;
    FieldType *field;
    ControlType *control;
    GraphicControlType *graphicControl;
    SliderControlType *sliderControl;
    ListType *list;
    TableType *table;
    FormLabelType *label;
    FormTitleType *title;
    FormPopupType *popup;
    FormGraffitiStateType *grfState;
    FormGadgetType *gadget;
    ScrollBarType *scrollBar;
} FormObjectType;
```

Your code should treat the `FormObjectType` structure as opaque. Do not attempt to change structure member values directly.

### Field Descriptions

- **ptr**
  - Used when the object’s type is not one of those specified below.

- **field**
  - Text field’s structure. See `FieldType`.

- **control**
  - Control’s structure. See `ControlType`.

- **graphicControl**
  - Graphic button structure. See `GraphicControlType`.

- **sliderControl**
  - Slider control structure. See `SliderControlType`.

---

**frmGadgetObj**

Gadget (custom object)

**frmScrollbarObj**

Scrollbar
list List object’s structure. See FormType.
table Table structure. See TableType.
bitmap Form bitmap’s structure. See FormBitmapType.
label Label’s structure. See FormLabelType.
title Form title’s structure. See FormTitleType.
popup Popup list’s structure. See FormPopupType.
grfState Graffiti shift indicator’s structure. See FrmGraffitiStateType.
gadget Gadget (custom UI resource) structure. See FormGadgetType.
scrollbar Scroll bar’s structure. See ScrollBarType.

Compatibility The graphicControl and sliderControl fields are only defined if 3.5 New Feature Set is present.

FormObjListType
The FormObjectListType structure specifies a user interface object that appears on the form.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the FormObjectListType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

typedef struct {
    FormObjectKind   objectType;
    UInt8            reserved;
    FormObjectType   object;
} FormObjListType;

Your code should treat the FormObjListType structure as opaque. Do not attempt to change structure member values directly.
Field Descriptions

objectType  Specifies the type of the object (control, field, etc.). See FormObjectKind.

reserved  Reserved for future use.

object  The C data structure that defines the object. See FormObjectType.

FormPopupType

The FormPopupType structure defines a popup list that appears on a form.

WARNING!  PalmSource, Inc. does not support or provide backward compatibility for the FormPopupType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.
typedef struct {
    UInt16      controlID;
    UInt16      listID;
} FormPopupType;

Your code should treat the FormPopupType structure as opaque. Do not attempt to change structure member values directly.

Field Descriptions

controlID    Resource ID of the popup trigger control that triggers the list’s display.

listID       Resource ID of the list object that defines the popup list.

FormPtr

The FormPtr type defines a pointer to a FormType structure.

typedef FormType *FormPtr;

FormRectangleType

The FormRectangleType structure defines a rectangle that appears on the form.

typedef struct {
    FormObjAttrType   attr;
    RectangleType     rect;
} FormRectangleType;

Your code should treat the FormRectangleType structure as opaque. Do not attempt to change structure member values directly.

Field Descriptions

attr          See FormObjAttrType.

rect          Location and size of the rectangle.
FormTitleType

The FormTitleType structure defines the title of the form.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the FormTitleType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

typedef struct {
    RectangleType  rect;
    char           *text;
} FormTitleType;

Your code should treat the FormTitleType structure as opaque. Do not attempt to change structure member values directly.

Field Descriptions

rect The location and size of the title area.

text Text of the title.

FormType

The FormType structure and supporting structures are defined below.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the FormType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

typedef struct {
    WindowType                window;
    UInt16                    formId;
    FormAttrType              attr;
    WinHandle                 bitsBehindForm;
    FormEventHandlerType      *handler;
} FormType;
### Forms

#### Form Data Structures

```c
UInt16 focus;
UInt16 defaultButton;
UInt16 helpRscId;
UInt16 menuRscId;
UInt16 numObjects;
FormObjListType *objects;
}
```

Your code should treat the `FormType` structure as opaque. Do not attempt to change structure member values directly.

#### Field Descriptions

- **window**: Structure of the window object that corresponds to the form. See `WindowType`. Access this field with `FrmGetWindowHandle`.
- **formId**: ID number of the form, specified by the application developer. This ID value is part of the event data of `frmOpenEvent`. The ID should match the form’s resource ID. Access this field with `FrmGetFormId`.
- **attr**: Form object attributes. See `FormAttrType`.
- **bitsBehindForm**: Used to save all the bits behind the form so the screen can be properly refreshed when the form is closed. This field is for internal use only by modal forms.
- **handler**: Routine called when the form needs to handle an event. You typically set this in your application’s event handling function by calling `FrmSetEventHandler`.
- **focus**: Index of a field or table object within the form that contains the focus. Any `KeyDownEvent` is passed to the object that has the focus. Set to `n0Focus` if no object has the focus. Set this field with `FrmSetFocus`.
The FrmGraffitiStateType structure defines the graffiti shift indicator.

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the FrmGraffitiStateType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```c
typedef struct{
    PointerType    pos;
}FrmGraffitiStateType;
```

Your code should treat the FrmGraffitiStateType structure as opaque. Do not attempt to change structure member values directly.

**Field Descriptions**

- **pos** Location of the graffiti shift indicator.
Form Constants

The following form constants are defined:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noFocus</td>
<td>0xffff</td>
<td>No form object has the focus</td>
</tr>
<tr>
<td>frmRedrawUpdateCode</td>
<td>0x8000</td>
<td>Indicates that the form should be redrawn; flag in a frmUpdateEvent.</td>
</tr>
<tr>
<td>frmNoSelectedControl</td>
<td>0xff</td>
<td>Returned by FrmGetControlGroupSelection if no control is selected.</td>
</tr>
<tr>
<td>frmResponseCreate</td>
<td>1974</td>
<td>Passed to FormCheckResponseFuncType to indicate that the function should perform initialization.</td>
</tr>
<tr>
<td>frmResponseQuit</td>
<td>0xBEEF</td>
<td>Passed to FormCheckResponseFuncType to indicate that the function should perform cleanup.</td>
</tr>
</tbody>
</table>

Form Resources

The following resources are associated with forms and with the objects on a form whose data structures are defined above:

- Form—Form Resource (tFRM)
- Alert dialog—Alert Resource (Talt)
- Bitmap—Form Bitmap Resource (tFBM)
- Button—Button Resource (tBTN)
- Check box—Check Box Resource (tCBX)
- Field—Field Resource (tFLD)
- Gadget (custom object)—Gadget Resource (tGDT)
- Graffiti shift indicator—Graffiti Shift Indicator Resource (tGSI)
- Label—Label Resource (tLBL)
Form Functions

FrmAlert

Purpose  Create a modal dialog from an alert resource and display it until the user selects a button in the dialog.

Declared In  Form.h

Prototype  UInt16 FrmAlert (UInt16 alertId)

Parameters  -> alertId  ID of the alert resource.

Result  Returns the item number of the button the user selected. A button’s item number is determined by its order in the alert dialog; the first button has the item number 0 (zero).

NOTE:  A default button press is simulated if the user switches to a different application while a modal dialog is active.

See Also  FrmDoDialog, FrmCustomAlert, FrmCustomResponseAlert
**FrmCloseAllForms**

**Purpose**
Send a `frmCloseEvent` to all open forms.

**Declared In**
Form.h

**Prototype**
```c
void FrmCloseAllForms (void)
```

**Parameters**
None.

**Result**
Returns nothing.

**Comments**
Applications can call this function to ensure that all forms are closed cleanly before exiting `PilotMain`; that is, before termination.

**See Also**
`FrmSaveAllForms`

---

**FrmCopyLabel**

**Purpose**
Copy the passed string into the data structure of the specified label object in the active form.

**Declared In**
Form.h

**Prototype**
```c
void FrmCopyLabel (FormType *formP, UInt16 labelID, const Char *newLabel)
```

**Parameters**
- `-> formP` Pointer to the form object (FormType structure).
- `-> labelID` ID of form label object.
- `-> newLabel` Pointer to a null-terminated string.

**Result**
Returns nothing.

**Comments**
The size of the new label **must not** exceed the size of the label defined in the resource. When defining the label in the resource, specify an initial size at least as big as any of the strings that will be
assigned dynamically. This function redraws the label if the form’s usable attribute and the label’s visible attribute are set.

This function redraws the label but does not erase the old one first. If the new label is shorter than the old one, the end of the old label will still be visible. To avoid this, you can hide the label using **FrmHideObject**, then show it using **FrmShowObject**, after using **FrmCopyLabel**.

Note that **FrmCopyLabel** copies the passed string into memory already allocated for the label. Thus, the string doesn’t need to remain in existence once **FrmCopyLabel** returns.

**See Also**  
**FrmGetLabel**

### FrmCopyTitle

**Purpose**  
Copy a new title over the form’s current title. If the form is visible, the new title is drawn.

**Declared In**  
**Form.h**

**Prototype**  
```c
void FrmCopyTitle (FormType *formP, 
                  const Char *newTitle)
```

**Parameters**  
- `formP`  
  Pointer to the form object (**FormType** structure).
- `newTitle`  
  Pointer to the new title string.

**Result**  
Returns nothing.

**Comments**  
The size of the new title must not exceed the title size defined in the resource. When defining the title in the resource, specify an initial size at least as big as any of the strings to be assigned dynamically.

**See Also**  
**FrmGetTitle**, **FrmSetTitle**
FrmCustomAlert

Purpose: Create a modal dialog from an alert resource and display the dialog until the user taps a button in the alert dialog.

Declared In: Form.h

Prototype:(UInt16 FrmCustomAlert (UInt16 alertId, const Char *s1, const Char *s2, const Char *s3))

Parameters:
- alertId: Resource ID of the alert.
- s1, s2, s3: Strings to replace ^1, ^2, and ^3 (see Comments).

Result: Returns the number of the button the user tapped (the first button is zero).

Comments: A button’s item number is determined by its order in the alert template; the first button has the item number zero.

Up to three strings can be passed to this routine. They are used to replace the variables ^1, ^2 and ^3 that are contained in the message string of the alert resource.

If the variables ^1, ^2, and ^3 occur in the message string, do not pass NULL for the arguments s1, s2, and s3. If you want an argument to be ignored, pass the empty string (""). In Palm OS 2.0 or below, pass a string containing a space (" ") instead of the empty string.

NOTE: A default button press is simulated if the user switches to a different application while a modal dialog is active.

See Also: FrmAlert, FrmDoDialog, FrmCustomResponseAlert
FrmCustomResponseAlert

**Purpose**
Create a modal dialog with a text field from an alert resource and display it until the user taps a button in the alert dialog.

**Declared In**
Form.h

**Prototype**
UInt16 FrmCustomResponseAlert (UInt16 alertId, const Char *s1, const Char *s2, const Char *s3, Char *entryStringBuf, Int16 entryStringBufLength, FormCheckResponseFuncPtr callback)

**Parameters**
- **alertId**
  Resource ID of the alert.
- **s1, s2, s3**
  Strings to replace ^1, ^2, and ^3. See the Comments in **FrmCustomAlert** for more information.
- **entryStringBuf**
  The string the user entered in the text field.
- **entryStringBufLength**
  The maximum length for the string in entryStringBuf.
- **callback**
  A callback function that processes the string. See **FormCheckResponseFuncType**. Pass NULL if there is no callback.

**Result**
Returns the number of the button the user tapped (the first button is zero).

**Comments**
This function differs from **FrmCustomAlert** in these ways:

- The dialog it displays contains a text field for user entry. The text that the user enters is returned in the entryStringBuf parameter.
- When the user taps a button, the callback function is called and is passed the button number and entryStringBuf. The dialog is only dismissed if the callback returns true. This behavior allows you to perform error checking on the
string that the user entered and give the user a chance to re-
enter the string.

The callback function is also called with special constants
when the alert dialog is being initialized and when it is being
deallocated. This allows the callback to perform any
necessary initialization and cleanup.

**NOTE:** A default button press is simulated if the user switches
to a different application while a modal dialog is active.

<table>
<thead>
<tr>
<th>Compatibility</th>
<th>Implemented only if 3.5 New Feature Set is present.</th>
</tr>
</thead>
<tbody>
<tr>
<td>See Also</td>
<td>FrmAlert, FrmDoDialog</td>
</tr>
</tbody>
</table>

## FrmDeleteForm

### Purpose
Release the memory occupied by a form. Any memory allocated to
objects in the form is also released.

### Declared In
Form.h

### Prototype
void FrmDeleteForm (FormType *formP)

### Parameters
-> formP Pointer to the form object (FormType structure).

### Result
Returns nothing.

### Comments
This function doesn’t modify the display.

### Compatibility
If 3.5 New Feature Set is present and the form contains an extended
gadget, this function calls the gadget’s callback with
formGadgetDeleteCmd. See FormGadgetHandlerType.

### See Also
FrmInitForm, FrmReturnToForm
**FrmDispatchEvent**

**Purpose**  Dispatch an event to the application’s handler for the form.

**Declared In**  Form.h

**Prototype**  Boolean FrmDispatchEvent (EventType *eventP)

**Parameters**  -> eventP  Pointer to an event.

**Result**  Returns the Boolean value returned by the form’s event handler or FrmHandleEvent. (If the form’s event handler returns false, the event is passed to FrmHandleEvent.) This function also returns false if the form specified in the event is invalid.

**Comments**  The event is dispatched to the current form’s handler unless the form ID is specified in the event data, as, for example, with frmOpenEvent or frmGotoEvent. A form’s event handler (FormEventHandlerType) is registered by FrmSetEventHandler.

Note that if the form does not have a registered event handler, this function causes a fatal error.

**FrmDoDialog**

**Purpose**  Display a modal dialog until the user taps a button in the dialog.

**Declared In**  Form.h

**Prototype**  UInt16 FrmDoDialog (FormType *formP)

**Parameters**  -> formP  Pointer to the form object (FormType structure).

**Result**  Returns the resource ID of the button the user tapped.
NOTE: A default button press is simulated if the user switches to a different application while a modal dialog is active.

Comments
Before calling FrmDoDialog you must have called FrmInitForm to load and initialize the dialog and you must have then set the event handler, if one is needed. After the call, read any values needed from the dialog’s objects and then call FrmDeleteForm to release the memory occupied by the dialog.

See Also FrmInitForm, FrmCustomAlert, FrmCustomResponseAlert

FrmDrawForm

Purpose
Draw all objects in a form and the frame around the form.

Declared In Form.h

Prototype
void FrmDrawForm (FormType *formP)

Parameters
-> formP Pointer to the form object (FormType structure).

Result
Returns nothing.

Comments
If the saveBehind form attribute is set and the form is visible, this function saves the bits behind the form using the bitsBehindForm field in the FormType structure.

You should call this function in response to a frmOpenEvent.

If you do any custom drawing, you should do so after you call this function not before. If you do custom drawing, respond to frmUpdateEvent as well as frmOpenEvent, and be sure to return true to specify that the frmUpdateEvent was handled. The default event handler for frmUpdateEvent calls FrmDrawForm, so if you allow the event to fall through by returning false, your custom drawing is erased.
Compatibility

If 3.5 New Feature Set is present, FrmDrawForm erases the form’s window before performing any drawing. Thus, it is especially important to do any custom drawing after this function call on Palm OS 3.5 and higher.

If 3.5 New Feature Set is present and the form contains an extended gadget, this function calls the gadget’s callback with formGadgetDrawCmd. See FormGadgetHandlerType.

See Also
FrmEraseForm, FrmInitForm

FrmEraseForm

Purpose
Erase a form from the display.

Declared In
Form.h

Prototype
void FrmEraseForm (FormType *formP)

Parameters
-> formP Pointer to the form object (FormType structure).

Result
Returns nothing.

Comments
If the region obscured by the form was saved by FrmDrawForm, this function restores that region.
**FrmGetActiveField**

**Purpose**  Return the active field for a specified form.

**Declared In**  Form.h

**Prototype**  
```
FieldType *FrmGetActiveField
(const FormType *formP)
```

**Parameters**  
- `-> formP`  Pointer to the form for which the active field should be returned, or NULL if the active field on the active form is desired.

**Result**  Returns a pointer to the field object of the active field, or NULL if the form doesn’t have an active field or if there is no active form.

**Comments**  This function will most often be called with a NULL parameter to obtain the active field on the active form.

**Compatibility**  Implemented only if 4.0 New Feature Set is present.

**See Also**  FrmGetActiveForm

**FrmGetActiveForm**

**Purpose**  Return the currently active form.

**Declared In**  Form.h

**Prototype**  
```
FormType *FrmGetActiveForm (void)
```

**Parameters**  None.

**Result**  Returns a pointer to the form object of the active form.

**Comments**  You should not call the FrmGetActiveForm function when a popup window is open. There is no active form while a popup is
Forms
Form Functions

displayed, and the value returned from FrmGetActiveForm in this situation has no meaning.

See Also  FrmGetActiveField, FrmGetActiveFormID, FrmSetActiveForm

**FrmGetActiveFormID**

**Purpose**  Return the ID of the currently active form.

**Declared In**  Form.h

**Prototype**  UInt16 FrmGetActiveFormID (void)

**Parameters**  None.

**Result**  Returns the active form’s ID number.

See Also  FrmGetActiveForm

**FrmGetControlGroupSelection**

**Purpose**  Return the item number of the control selected in a group of controls.

**Declared In**  Form.h

**Prototype**  UInt16 FrmGetControlGroupSelection (const FormType *formP, UInt8 groupNum)

**Parameters**  -> formP  Pointer to the form object (FormType structure).

-> groupNum  Control group number.

**Result**  Returns the item number of the selected control; returns frmNoSelectedControl if no item is selected.
Comments

The item number is the index into the form object's data structure.

**NOTE:** `FrmSetControlGroupSelection` sets the selection in a control group based on an object ID, **not** its index, which `FrmGetControlGroupSelection` returns.

Compatibility

On versions prior to 3.5, this function returned a `Byte` instead of `UInt16`.

See Also  
`FrmGetObjectId`, `FrmGetObjectPtr`,  
`FrmSetControlGroupSelection`

---

**FrmGetControlValue**

Purpose

Return the current value of a control.

Declared In

`Form.h`

Prototype

```c
Int16 FrmGetControlValue (const FormType *formP, UInt16 objIndex)
```

Parameters

- `-> formP` Pointer to the form object (`FormType` structure).
- `-> objIndex` Index of the control object in the form object's data structure. You can obtain this by using `FrmGetObjectIndex`.

Result

Returns the current value of the control. For most controls the return value is either 0 (off) or 1 (on). For sliders, this function returns the value of the `value` field.

Comments

The caller must specify a valid index. This function is valid only for push button and check box control objects.

See Also  
`FrmSetControlValue`
FrmGetFirstForm

**Purpose**
Return the first form in the window list.

**Declared In**
Form.h

**Prototype**
FormType *FrmGetFirstForm (void)

**Parameters**
None.

**Result**
Returns a pointer to a form object, or NULL if there are no forms.

**Comments**
The window list is a LIFO stack. The last window created is the first window in the window list.

FrmGetFocus

**Purpose**
Return the item (index) number of the object that has the focus.

**Declared In**
Form.h

**Prototype**
UInt16 FrmGetFocus (const FormType *formP)

**Parameters**
-> formP Pointer to the form object (FormType structure).

**Result**
Returns the index of the object (UI element) that has the focus, or returns noFocus if none does. To convert the object index to an ID, use FrmGetObjectId.

**See Also**
FrmGetObjectPtr, FrmSetFocus
**FrmGetFormBounds**

**Purpose**
Return the visual bounds of the form; the region returned includes the form’s frame.

**Declared In**
Form.h

**Prototype**
```c
void FrmGetFormBounds (const FormType *formP,
RectangleType *rP)
```

**Parameters**
- `-> formP` Pointer to the form object (FormType structure).
- `<- rP` Pointer to a RectangleType structure where the bounds is returned.

**Result**
Returns nothing. The bounds of the form are returned in `r`.

**FrmGetFormId**

**Purpose**
Return the resource ID of a form.

**Declared In**
Form.h

**Prototype**
```c
UInt16 FrmGetFormId (const FormType *formP)
```

**Parameters**
- `-> formP` Pointer to the form object (FormType structure).

**Result**
Returns form resource ID.

**See Also**
FrmGetFormPtr
**FrmGetFormPtr**

**Purpose**
Return a pointer to the form that has the specified ID.

**Declared In**
Form.h

**Prototype**
FormType *FrmGetFormPtr (UInt16 formId)

**Parameters**
- `formId` Form ID number.

**Result**
Returns a pointer to the form object, or NULL if the form is not in memory.

**See Also**
FrmGetFormId

**FrmGetGadgetData**

**Purpose**
Return the value stored in the data field of the gadget object.

**Declared In**
Form.h

**Prototype**
void *FrmGetGadgetData (const FormType *formP, UInt16 objIndex)

**Parameters**
- `formP` Pointer to the form object (FormType structure).
- `objIndex` Index of the gadget object in the form object’s data structure. You can obtain this by using FrmGetObjectIndex.

**Result**
Returns a pointer to the custom gadget’s data.

**Comments**
Gadget objects provide a way for an application to attach custom gadgetry to a form. In general, the data field of a gadget object contains a pointer to the custom object’s data structure.

**See Also**
FrmSetGadgetData, FrmSetGadgetHandler
FrmGetLabel

Purpose
Return pointer to the text of the specified label object in the specified form.

Declared In
Form.h

Prototype
const Char *FrmGetLabel (const FormType *formP, UInt16 labelID)

Parameters
- > formP Pointer to the form object (FormType structure).
- > labelID ID of the label object.

Result
Returns a pointer to the label string.

Comments
Does not make a copy of the string; returns a pointer to the string. The object must be a label.

See Also
FrmCopyLabel

FrmGetNumberOfObjects

Purpose
Return the number of objects in a form.

Declared In
Form.h

Prototype
UInt16 FrmGetNumberOfObjects (const FormType *formP)

Parameters
- > formP Pointer to the form object (FormType structure).

Result
Returns the number of objects in the specified form.

See Also
FrmGetObjectPtr, FrmGetObjectId
FrmGetObjectBounds

**Purpose**
Retrieve the bounds of an object given its form and index.

**Declared In**
Form.h

**Prototype**
void FrmGetObjectBounds (const FormType *formP, Uint16 objIndex, RectangleType *rP)

**Parameters**
- formP: Pointer to the form object (FormType structure).
- objIndex: Index of an object in the form. You can obtain this by using FrmGetObjectIndex.
- rP: Pointer to a RectangleType structure where the object bounds are returned. The bounds are in window-relative coordinates.

**Result**
Returns nothing. The object’s bounds are returned in r.

**See Also**
FrmGetObjectPosition, FrmSetObjectPosition

FrmGetObjectId

**Purpose**
Return the ID of the specified object.

**Declared In**
Form.h

**Prototype**
UInt16 FrmGetObjectId (const FormType *formP, Uint16 objIndex)

**Parameters**
- formP: Pointer to the form object (FormType structure).
Forms
Form Functions

-> objIndex  Index of an object in the form. You can obtain this by using FrmGetObjectIndex.

Result  Returns the ID number of an object or frmInvalidObjectId if the objIndex parameter is invalid.

See Also  FrmGetObjectPtr

FrmGetObjectIndex

Purpose  Return the index of an object in the form’s objects list.

Declared In  Form.h

Prototype  UInt16 FrmGetObjectIndex (const FormType *formP, UInt16 objID)

Parameters  -> formP  Pointer to the form object (FormType structure).

-> objID  ID of an object in the form.

Result  Returns the index of the specified object (the index of the first object is 0), or frmInvalidObjectId if the supplied object ID is invalid.

Comments  Bitmaps use a different mechanism for IDs than the rest of the form objects. When finding a bitmap with FrmGetObjectIndex, you need to pass the bitmap’s resource ID, not the ID of the form bitmap object. (Passing the ID of the form bitmap object may or may not give you the right object back, depending on how you created the objects.)

This means that if you’ve got the same bitmap in two different form bitmap objects on the same form, you won’t be able to use FrmGetObjectIndex to get at the second one; it’ll always return the first.

See Also  FrmGetObjectPtr, FrmGetObjectId


FrmGetObjectIndexFromPtr

**Purpose**
Return an object’s index.

**Declared In**
Form.h

**Prototype**

```c
UInt16 FrmGetObjectIndexFromPtr
(const FormType *formP, void *objP)
```

**Parameters**
- `formP` Pointer to a `FormType`.
- `objP` Pointer to an object.

**Result**
Returns the object’s index. `frmInvalidObjectId` is returned if `objP` is not associated with the form.

**Comments**

Many form functions (FrmGetObjectIndex, FrmHideObject, and FrmGetObjectBounds, for example) take an object index as one of their arguments. The most common way to get an object’s index is to call FrmGetObjectIndex. FrmGetObjectIndex takes a form ID and returns the form object’s index. This is the routine one should use in most cases, because the application usually knows the object ID. However, gadgets and specifically extended gadgets, have APIs with callbacks that pass back the gadget pointer and not the ID. In those cases, the only way to get the object index (so one can use the FrmGetObject* APIs) is to use FrmGetObjectIndexFromPtr.

If you need the same functionality on pre-Palm OS 4.0 systems then you can accomplish the same thing with the following code snippet.

```c
UInt16 index;
UInt16 objIndex = frmInvalidObjectId;
UInt16 numObjects = FrmGetNumberOfObjects(frmP)
for (index = 0; index < numObjects; index++) {
    if (FrmGetObjectPtr(index) == myObjPtr) {
        // Found it
        objIndex = index;
        break;
    }
}
```
**Compatibility**

Implemented only if [4.0 New Feature Set](#) is present.

---

**FrmGetObjectPosition**

**Purpose**

Return the coordinates of the specified object relative to the form.

**Declared In**

Form.h

**Prototype**

```c
void FrmGetObjectPosition (const FormType *formP,
UInt16 objIndex, Coord *x, Coord *y)
```

**Parameters**

- `-> formP`  
  Pointer to the form object ([FormType](#) structure).

- `-> objIndex`  
  Index of an object in the form. You can obtain this by using [FrmGetObjectIndex](#).

- `<- x, y`  
  Pointers where the window-relative x and y positions of the object are returned. These locate the top-left corner of the object.

**Result**

Returns nothing.

**See Also**  
[FrmGetObjectBounds](#), [FrmSetObjectPosition](#)

---

**FrmGetObjectPtr**

**Purpose**

Return a pointer to the data structure of an object in a form.

**Declared In**

Form.h

**Prototype**

```c
void *FrmGetObjectPtr (const FormType *formP,
UInt16 objIndex)
```

**Parameters**

- `-> formP`  
  Pointer to the form object ([FormType](#) structure).
-> objIndex  

Index of an object in the form. You can obtain this by using FrmGetObjectIndex.

Result  

Returns a pointer to an object in the form.

See Also  

FrmGetObjectId

FrmGetObjectType

Purpose  

Return the type of an object.

Declared In  

Form.h

Prototype  

FormObjectKind FrmGetObjectType  
(const FormType *formP, UInt16 objIndex)

Parameters  

-> formP  

Pointer to the form object (FormType structure).

-> objIndex  

Index of an object in the form. You can obtain this by using FrmGetObjectIndex.

Result  

Returns FormObjectKind of the item specified. See FormObjectKind.

FrmGetTitle

Purpose  

Return a pointer to the title string of a form.

Declared In  

Form.h

Prototype  

const Char *FrmGetTitle (const FormType *formP)

Parameters  

-> formP  

Pointer to the form object (FormType structure).

Result  

Returns a pointer to title string, or NULL if there is no title string or there is an error finding it.
Comments 
This is a pointer to the internal structure itself, not to a copy.

See Also 
FrmCopyTitle, FrmSetTitle

FrmGetWindowHandle

Purpose 
Return the window handle of a form.

Declared In 
Form.h

Prototype 
WinHandle FrmGetWindowHandle (const FormType *formP)

Parameters 
-> formP Pointer to the form object (FormType structure).

Result 
Returns the handle of the memory block that contains the form data structure. Since the form structure begins with the WindowType, this is also a WinHandle.

FrmGotoForm

Purpose 
Send a frmCloseEvent to the current form; send a frmLoadEvent and a frmOpenEvent to the specified form.

Declared In 
Form.h

Prototype 
void FrmGotoForm (UInt16 formId)

Parameters 
-> formId ID of the form to display.

Result 
Returns nothing.

Comments 
The default form event handler (FrmHandleEvent) erases and disposes of a form when it receives a frmCloseEvent.

See Also 
FrmPopupForm
**FrmHandleEvent**

**Purpose**  
Handle the event that has occurred in the form.

**Declared In**  
Form.h

**Prototype**  
Boolean FrmHandleEvent (FormType *formP, EventType *eventP)

**Parameters**
- **formP**  
Pointer to the form object (FormType structure).
- **eventP**  
Pointer to the event data structure (EventType).

**Result**  
Returns true if the event was handled.

**Comments**  
Never call this function directly. Call FrmDispatchEvent instead. FrmDispatchEvent passes events to a form’s custom event handler and then, if the event was not handled, to this function.

---

**WARNING!** You should never call this function directly. You should call the FrmDispatchEvent function instead.

---

Table 11.1 provides an overview of how FrmHandleEvent handles different events.
Table 11.1 FrmHandleEvent Actions

<table>
<thead>
<tr>
<th>When FrmHandleEvent receives...</th>
<th>FrmHandleEvent performs these actions...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ctlEnterEvent</strong></td>
<td>Passes the event and a pointer to the object the event occurred in to <strong>CtlHandleEvent</strong>. The object pointer is obtained from the event data. If the control is part of an exclusive control group, it deselects the currently selected control of the group first.</td>
</tr>
<tr>
<td><strong>ctlRepeatEvent</strong></td>
<td>Passes the event and a pointer to the object the event occurred in to <strong>CtlHandleEvent</strong>. The object pointer is obtained from the event data.</td>
</tr>
<tr>
<td><strong>ctlSelectEvent</strong></td>
<td>Checks if the control is a Popup Trigger Control. If it is, the list associated with the popup trigger is displayed until the user makes a selection or touches the pen outside the bounds of the list. If a selection is made, a <strong>popSelectEvent</strong> is added to the event queue.</td>
</tr>
<tr>
<td><strong>fldEnterEvent or fldHeightChangedEvent</strong></td>
<td>Checks if a field object or a table object has the focus and passes the event to the appropriate handler (<strong>FldHandleEvent</strong> or <strong>TblHandleEvent</strong>). The table object is also a container object, which may contain a field object. If <strong>TblHandleEvent</strong> receives a field event, it passes the event to the field object contained within it.</td>
</tr>
<tr>
<td><strong>frmCloseEvent</strong></td>
<td>Erases the form and releases any memory allocated for it.</td>
</tr>
<tr>
<td><strong>frmGadgetEnterEvent</strong></td>
<td>Passes the event to the gadget’s callback function if the gadget has one. See <strong>FormGadgetHandlerType</strong>.</td>
</tr>
<tr>
<td><strong>frmGadgetMiscEvent</strong></td>
<td>Passes the event to the gadget’s callback function if the gadget has one. See <strong>FormGadgetHandlerType</strong>.</td>
</tr>
<tr>
<td><strong>frmTitleEnterEvent</strong></td>
<td>Tracks the pen until it is lifted. If it is lifted within the bounds of the form title, adds a <strong>frmTitleSelectEvent</strong> event to the event queue.</td>
</tr>
</tbody>
</table>
### Table 11.1 FrmHandleEvent Actions (continued)

<table>
<thead>
<tr>
<th>When FrmHandleEvent receives...</th>
<th>FrmHandleEvent performs these actions...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>frmTitleSelectEvent</strong></td>
<td>Adds a <strong>keyDownEvent</strong> with the <code>vchrMenu</code> character to the event queue.</td>
</tr>
<tr>
<td><strong>frmUpdateEvent</strong></td>
<td>Calls <strong>FrmDrawForm</strong> to redraw the form.</td>
</tr>
<tr>
<td><strong>keyDownEvent</strong></td>
<td>Passes the event to the handler for the object that has the focus. If no object has the focus, the event is ignored.</td>
</tr>
<tr>
<td><strong>lstEnterEvent</strong></td>
<td>Passes the event and a pointer to the object the event occurred in to <strong>LstHandleEvent</strong>. The object pointer is obtained from the event data.</td>
</tr>
<tr>
<td><strong>menuCmdBarOpenEvent</strong></td>
<td>Checks if a field object or a table object has the focus and passes the event to the appropriate handler (FldHandleEvent or TblHandleEvent), broadcasts the notification <code>sysNotifyMenuCmdBarOpenEvent</code>, and then displays the command toolbar.</td>
</tr>
<tr>
<td><strong>menuEvent</strong></td>
<td>Checks if the menu command is one of the system edit menu commands. The system provides a standard edit menu that contains the commands Undo, Cut, Copy, Paste, Select All, and Keyboard. <strong>FrmHandleEvent</strong> responds to these commands.</td>
</tr>
<tr>
<td><strong>penDownEvent</strong>; pen position in the bounds of the form object</td>
<td>Checks the list of objects contained by the form to determine if the pen is within the bounds of one. If it is, the appropriate handler is called to handle the event, for example, if the pen is in a control, <strong>CtlHandleEvent</strong> is called. If the pen isn’t within the bounds of an object, the event is ignored by the form. If the pen is within the bounds of the help icon, it is tracked until it is lifted, and if it’s still within the help icon bounds, the help dialog is displayed.</td>
</tr>
<tr>
<td><strong>popSelectEvent</strong></td>
<td>Sets the label of the popup trigger to the current selection of the popup list.</td>
</tr>
</tbody>
</table>
Table 11.1 FrmHandleEvent Actions (continued)

<table>
<thead>
<tr>
<th>When FrmHandleEvent receives...</th>
<th>FrmHandleEvent performs these actions...</th>
</tr>
</thead>
<tbody>
<tr>
<td>sclEnterEvent or</td>
<td>Passes the event and a pointer to the object the event occurred in to SclHandleEvent.</td>
</tr>
<tr>
<td>sclRepeatEvent</td>
<td></td>
</tr>
<tr>
<td>tblEnterEvent</td>
<td>Passes the event and a pointer to the object the event occurred in to TblHandleEvent. The object pointer is obtained from the event data.</td>
</tr>
</tbody>
</table>

Compatibility

FrmHandleEvent only handles frmTitleSelectEvent, menuCmdBarOpenEvent, frmGadgetEnterEvent, and frmGadgetMiscEvent if 3.5 New Feature Set is present. If 5.0 New Feature Set is present, this function should be considered “System Use Only”; applications should do what they can to avoid using it.

See Also

FrmDispatchEvent

FrmHelp

Purpose

Display the specified help message until the user taps the Done button in the help dialog.

Declared In

Form.h

Prototype

void FrmHelp (UInt16 helpMsgId)

Parameters

-> helpMsgId Resource ID of help message string.

Result

Returns nothing.

Comments

The help message is displayed in a modal dialog that supports scrolling the text if necessary.
FrmHideObject

**Purpose**
Erase the specified object and set its attribute data (usable bit) so that it does not redraw or respond to the pen.

**Declared In**
Form.h

**Prototype**
```c
void FrmHideObject (FormType *formP, UInt16 objIndex)
```

**Parameters**
- `formP` Pointer to the form object (`FormType` structure).
- `objIndex` Index of an object in the form. You can obtain this by using `FrmGetObjectIndex`.

**Result**
Returns nothing.

**Compatibility**
Prior to OS version 3.2, this function did not set the usable bit of the object attribute data to false. On an OS version prior to 3.2 you can work around this bug by directly setting this bit to false yourself.

On versions of Palm OS prior to 3.5 this function doesn’t affect lists or tables. On Palm OS 3.5 it operates correctly on lists but doesn’t have any effect on tables. On Palm OS 4.0 it operates correctly on both lists and tables.

If 3.5 New Feature Set is present and the object is an extended gadget, this function calls the gadget’s callback with formGadgetEraseCmd. See `FormGadgetHandlerType`.

**See Also**
FrmShowObject
**FrmInitForm**

**Purpose**  
Load and initialize a form resource.

**Declared In**  
Form.h

**Prototype**  
FormType *FrmInitForm (UInt16 rscID)

**Parameters**  
- `rscID`  
Resource ID of the form.

**Result**  
Returns a pointer to the form data structure.

When using debug ROMs, FrmInitForm displays an error message if the form has already been initialized.

**Comments**  
This function does not affect the display (use FrmDrawForm to draw the form) nor make the form active (use FrmSetActiveForm to make it active).

For each initialized form, you must call FrmDeleteForm to release the form memory when you are done with the form. Alternatively, you can free the active form by calling FrmReturnToForm.

**See Also**  
FrmDoDialog, FrmDeleteForm, FrmReturnToForm
**FrmNewBitmap**

**Purpose**  Create a new form bitmap dynamically.

**Declared In**  Form.h

**Prototype**  

```
FormBitmapType *FrmNewBitmap (FormType **formPP,
UInt16 ID, UInt16 rscID, Coord x, Coord y)
```

**Parameters**

- `<-> formPP`  Pointer to a pointer to the form in which the new bitmap is installed. This value is not a handle; that is, the old `formPP` value is not necessarily valid after this function returns because the form may be moved in memory. In subsequent calls, always use the new `formPP` value returned by this function.
- `ID`  Symbolic ID of the bitmap, specified by the developer. By convention, this ID should match the resource ID (not mandatory).
- `rscID`  Numeric value identifying the resource that provides the bitmap. This value must be unique within the application scope.
- `x`  Horizontal coordinate of the upper-left corner of the bitmap’s boundaries, relative to the window in which it appears.
- `y`  Vertical coordinate of the upper-left corner of the bitmap’s boundaries, relative to the window in which it appears.

**Result**  Returns a pointer to the new bitmap, or 0 if the call did not succeed. The most common cause of failure is lack of memory.

**Compatibility**  Implemented only if **3.0 New Feature Set** is present.

**See Also**  [FrmRemoveObject](#)
FrmNewForm

**Purpose**  
Create a new form object dynamically.

**Declared In**  
Form.h

**Prototype**  
FormType *FrmNewForm (UInt16 formID,  
const Char *titleStrP, Coord x, Coord y,  
Coord width, Coord height, Boolean modal,  
UInt16 defaultButton, UInt16 helpRscID,  
UInt16 menuRscID)

**Parameters**

- `-> formID`  
  Symbolic ID of the form, specified by the developer. By convention, this ID should match the resource ID (not mandatory).

- `-> titleStrP`  
  Pointer to a string that is the title of the form.

- `-> x`  
  Horizontal coordinate of the upper-left corner of the form’s boundaries, relative to the window in which it appears.

- `-> y`  
  Vertical coordinate of the upper-left corner of the form’s boundaries, relative to the window in which it appears.

- `-> width`  
  Width of the form, expressed in pixels. Valid values are 1 - 160.

- `-> height`  
  Height of the form, expressed in pixels. Valid values are 1 - 160.

- `-> modal`  
  `true` specifies that the form ignores pen events outside its boundaries.

- `-> defaultButton`  
  Symbolic ID of the button that provides the form’s default action, specified by the developer.

- `-> helpRscID`  
  Symbolic ID of the resource that provides the form’s online help, specified by the developer. Only modal dialogs can have help resources.
Forms
Form Functions

-> menuRscID Symbolic ID of the resource that provides the form’s menus, specified by the developer.

**Result**
Returns a pointer to the new form object, or 0 if the call did not succeed. The most common cause of failure is lack of memory.

**Compatibility**
Implemented only if **3.0 New Feature Set** is present.

**See Also**
FrmValidatePtr, WinValidateHandle, FrmRemoveObject

### FrmNewGadget

**Purpose**
Create a new gadget dynamically and install it in the specified form.

**Declared In**
Form.h

**Prototype**
FormGadgetType *FrmNewGadget (FormType **formPP, UInt16 id, Coord x, Coord y, Coord width, Coord height)

**Parameters**

- **<> formPP** Pointer to a pointer to the form in which the new gadget is installed. This value is not a handle; that is, the old formPP value is not necessarily valid after this function returns because the form may be moved in memory. In subsequent calls, always use the new formPP value returned by this function.

- **-> id** Symbolic ID of the gadget, specified by the developer. By convention, this ID should match the resource ID (not mandatory).

- **-> x** Horizontal coordinate of the upper-left corner of the gadget’s boundaries, relative to the window in which it appears.

- **-> y** Vertical coordinate of the upper-left corner of the gadget’s boundaries, relative to the window in which it appears.
-> width  
Width of the gadget, expressed in pixels. Valid values are 1 - 160.

-> height  
Height of the gadget, expressed in pixels. Valid values are 1 - 160.

**Result**  
Returns a pointer to the new gadget object or 0 if the call did not succeed. The most common cause of failure is lack of memory.

**Comments**  
A gadget is a custom user interface object. For more information, see “Gadget Resource” on page 50.

**Compatibility**  
Implemented only if 3.0 New Feature Set is present.

**See Also**  
FrmRemoveObject

**FrmNewGsi**

**Purpose**  
Create a new Graffiti shift indicator dynamically and install it in the specified form.

**Declared In**  
Form.h

**Prototype**  
FrmGraffitiStateType *FrmNewGsi  
(FormType **formPP, Coord x, Coord y)

**Parameters**  
<-> formPP  
Pointer to a pointer to the form in which the new Graffiti shift indicator is installed. This value is not a handle; that is, the old formPP value is not necessarily valid after this function returns because the form may be moved in memory. In subsequent calls, always use the new formPP value returned by this function.

-> x  
Horizontal coordinate of the upper-left corner of the Graffiti shift indicator’s boundaries, relative to the window in which it appears.
-> y

Vertical coordinate of the upper-left corner of the Graffiti shift indicator’s boundaries, relative to the window in which it appears.

**Result**

Returns a pointer to the new gadget object or 0 if the call did not succeed. The most common cause of failure is lack of memory.

**Comments**

In normal operation, the Graffiti shift indicator is drawn in the lower-right portion of the screen when the user enters the shift keystroke. You use this function if the Graffiti shift indicator needs to be drawn in a nonstandard location. For example, the form manager uses it to draw the shift indicator in a custom alert dialog that contains a text field (**FrmCustomResponseAlert**).

**Compatibility**

Implemented only if **3.5 New Feature Set** is present.

**See Also**

**FrmRemoveObject**

---

**FrmNewLabel**

**Purpose**

Create a new label object dynamically and install it in the specified form.

**Declared In**

Form.h

**Prototype**

`FormLabelType *FrmNewLabel (FormType **formPP, UInt16 ID, const Char *textP, Coord x, Coord y, FontID font)`

**Parameters**

<-> formPP

Pointer to a pointer to the form in which the new label is installed. This value is not a handle; that is, the old formPP value is not necessarily valid after this function returns because the form may be moved in memory. In subsequent calls, always use the new formPP value returned by this function.
-> ID Symbolic ID of the label, specified by the developer. By convention, this ID should match the resource ID (not mandatory).

-> textP Pointer to a string that provides the label text. This string is copied into the label structure.

-> x Horizontal coordinate of the upper-left corner of the label’s boundaries, relative to the window in which it appears.

-> y Vertical coordinate of the upper-left corner of the label’s boundaries, relative to the window in which it appears.

-> font Font with which to draw the label text.

Result Returns a pointer to the new label object or 0 if the call did not succeed. The most common cause of failure is lack of memory.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also CtlValidatePointer, Frm removeObject

FrmPointInTitle

Purpose Check if a coordinate is within the bounds of the form’s title.

Declared In Form.h

Prototype Boolean FrmPointInTitle (const FormType *formP, Coord x, Coord y)

Parameters -> formP Pointer to the form object (FormType structure).

-> x, y Window-relative x and y coordinates.

Result Returns true if the specified coordinate is in the form’s title.

Compatibility Implemented only if 2.0 New Feature Set is present.
**FrmPopupForm**

**Purpose** Queues a `frmLoadEvent` and a `frmOpenEvent` for the specified form.

**Declared In** Form.h

**Prototype**

```c
void FrmPopupForm (UInt16 formId);
```

**Parameters**

- `formID` Resource ID of form to open.

**Result**

Returns nothing.

**Comments** This routine differs from `FrmGotoForm` in that the current form is not closed. You can call `FrmReturnToForm` to close a form opened by `FrmPopupForm`.

**FrmRemoveObject**

**Purpose** Remove the specified object from the specified form.

**Declared In** Form.h

**Prototype**

```c
Err FrmRemoveObject (FormType **formPP, UInt16 objIndex);
```

**Parameters**

- `formPP` Pointer to a pointer to the form from which this function removes an object. This value is not a handle; that is, the old `formPP` value is not necessarily valid after this function returns. In subsequent calls, always use the new `formPP` value returned by this function.

- `objIndex` The object to remove, specified as an index into the list of objects installed in the form. You can use the `FrmGetObjectIndex` function to discover this value.

**Result**

Returns 0 if no error.
You can use this function to remove any form object (a bitmap, control, list, and so on) and free the memory allocated to it within the form data structure. The data structures for most form objects are embedded within the form data structure memory chunk. This function frees that memory and moves the other objects, if necessary, to close up the memory “hole” and decrease the size of the form chunk.

Note that this function does not free memory outside the form data structure that may be allocated to an object, such as the memory allocated to the string in an editable field object.

Implemented only if 3.0 New Feature Set is present.

See Also FrmNewBitmap, FrmNewForm, FrmNewGadget, FrmNewLabel, CtlNewControl, FldNewField, LstNewList

**FrmRestoreActiveState**

Macro that restores the active window and form state.

Declared In Form.h

Prototype FrmRestoreActiveState (stateP)

Parameters -> stateP A pointer to the FormActiveStateType structure that you passed to FrmSaveActiveState when you saved the state.

Result Returns zero on success.

Comments Use this function to restore the state of displayed forms to the state that existed before you dynamically showed a new modal form. You must have previously called FrmSaveActiveState to save the state.

Implemented only if 3.0 New Feature Set is present.
**FrmReturnToForm**

**Purpose**  
Erase and delete the currently active form and make the specified form the active form.

**Declared In**  
Form.h

**Prototype**  
`void FrmReturnToForm (UInt16 formId)`

**Parameters**  
`-> formID`  
Resource ID of the form to return to.

**Result**  
Returns nothing.

**Comments**  
It is assumed that the form being returned to is already loaded into memory and initialized. Passing a form ID of 0 returns to the first form in the window list, which is the last form to be loaded.

FrmReturnToForm does not generate a frmCloseEvent when called from a modal form’s event handler. It assumes that you have already handled cleaning up your form’s variables since you are explicitly calling FrmReturnToForm.

**See Also**  
FrmGotoForm, FrmPopupForm
**FrmSaveActiveState**

**Purpose**  
Macro that saves the active window and form state.

**Declared In**  
Form.h

**Prototype**  
FrmSaveActiveState (stateP)

**Parameters**  
<-> stateP  
A pointer to a FormActiveStateType structure that is used to save the state. Pass the same pointer to FrmRestoreActiveState to restore the state. Treat the structure like a black box; that is, don’t attempt to read it or write to it.

**Result**  
Returns zero on success.

**Comments**  
Use this function to save the state of displayed forms before dynamically showing a new modal form. Call FrmRestoreActiveState to restore the state after you remove the modal form.

**Compatibility**  
Implemented only if 3.0 New Feature Set is present.

**FrmSaveAllForms**

**Purpose**  
Send a frmSaveEvent to all open forms.

**Declared In**  
Form.h

**Prototype**  
void FrmSaveAllForms (void)

**Parameters**  
None.

**Result**  
Returns nothing.

**See Also**  
FrmCloseAllForms
**FrmSetActiveForm**

**Purpose**
Set the active form. All input (key and pen) is directed to the active form and all drawing occurs there.

**Declared In**
Form.h

**Prototype**
void FrmSetActiveForm (FormType *formP)

**Parameters**
- `formP` Pointer to the form object (FormType structure).

**Result**
Returns nothing.

**Comments**
A penDownEvent outside the form but within the display area is ignored.

**Compatibility**
In Palm OS releases earlier than 3.5, this function generated a winEnterEvent for the new form immediately following the winExitEvent for the old form. Starting in Palm OS 3.5, FrmSetActiveForm does not generate the winEnterEvent. The winEnterEvent does not occur until the newly active form is drawn.

**See Also**
FrmGetActiveForm

**FrmSetCategoryLabel**

**Purpose**
Set the category label displayed on the title line of a form. If the form's visible attribute is set, redraw the label.

**Declared In**
Form.h

**Prototype**
void FrmSetCategoryLabel (const FormType *formP, Uint16 objIndex, Char *newLabel)

**Parameters**
- `formP` Pointer to the form object (FormType structure).
-> objIndex       Index of an object in the form. You can obtain this by using `FrmGetObjectIndex`.
-> newLabel       Pointer to the name of the new category.

**Result**       Returns nothing.

**Comments**     The pointer to the new label (newLabel) is saved in the object.

---

**FrmSetControlGroupSelection**

**Purpose**       Set the selected control in a group of controls.

**Declared In**   Form.h

**Prototype**     
```c
void FrmSetControlGroupSelection
        (const FormType *formP, UInt8 groupNum, UInt16 controlID)
```

**Parameters**    
- `-> formP`       Pointer to the form object (`FormType` structure).
- `-> groupNum`    Control group number.
- `-> controlID`   ID of control to set.

**Result**        Returns nothing.

**Comments**      This function unsets all the other controls in the group. The display is updated.

**NOTE:**         `FrmGetControlGroupSelection` returns the selection in a control group as an object index, **not** as an object ID, which `FrmSetControlGroupSelection` uses to set the selection.

**See Also**      `FrmGetControlGroupSelection`
**FrmSetControlValue**

**Purpose**  Set the current value of a control. If the control is visible, it’s redrawn.

**Declared In**  Form.h

**Prototype**  
```c
void FrmSetControlValue (const FormType *formP, UInt16 objIndex, Int16 newValue)
```

**Parameters**
- `formP`  Pointer to the form object (FormType structure).
- `objIndex`  Index of the control in the form. You can obtain this by using FrmGetObjectIndex.
- `newValue`  New value to set for the control. For sliders, specify a value between the slider’s minimum and maximum. For graphical controls, push buttons, or check boxes, specify 0 for off, nonzero for on.

**Result**  Returns nothing.

**Comments**  This function works only with graphical controls, sliders, push buttons, and check boxes. If you set the value of any other type of control, the behavior is undefined.

**See Also**  FrmGetControlValue
**FrmSetEventHandler**

**Purpose**
Registers the event handler callback routine for the specified form.

**Declared In**
Form.h

**Prototype**
```c
void FrmSetEventHandler (FormType *formP, FormEventHandlerType *handler)
```

**Parameters**
- `formP` Pointer to the form object (`FormType` structure).
- `handler` Address of the form event handler function, `FormEventHandlerType`.

**Result**
Returns nothing.

**Comments**
- `FrmDispatchEvent` calls this handler whenever it receives an event for a specific form.

FrmSetEventHandler must be called right after a form resource is loaded. The callback routine it registers is the mechanism for dispatching events to an application. The tutorial explains how to use callback routines.

**FrmSetFocus**

**Purpose**
Set the focus of a form to the specified object.

**Declared In**
Form.h

**Prototype**
```c
void FrmSetFocus (FormType *formP, UInt16 fieldIndex)
```

**Parameters**
- `formP` Pointer to the form object (`FormType` structure).
-> fieldIndex  Index of the object to get the focus in the form. You can obtain this by using FrmGetObjectIndex. You can pass the constant noFocus so that no object has the focus.

Result  Returns nothing.

Comments  You can set the focus to a field or table object. If the focus is set to a field object, this function turns on the insertion point in the field by calling FldGrabFocus internally.

See Also  FrmGetFocus

### FrmSetGadgetData

**Purpose**  Store a data value in the data field of the gadget object.

**Declared In**  Form.h

**Prototype**  void FrmSetGadgetData (FormType *formP, UInt16 objIndex, const void *data)

**Parameters**
- -> formP  Pointer to the form object (FormType structure).
- -> objIndex  Index of an object in the form. You can obtain this by using FrmGetObjectIndex.
- -> data  Application-defined value. This value is stored into the data field of the gadget data structure (FormGadgetType).

Result  Returns nothing.

Comments  Gadget objects provide a way for an application to attach custom gadgetry to a form. Typically, the data field of a gadget object contains a pointer to the custom object’s data structure.

See Also  FrmGetGadgetData, FrmSetGadgetHandler
FrmSetGadgetHandler

Purpose
Registers the gadget event handler callback routine for the specified gadget on the specified form.

Declared In
Form.h

Prototype
void FrmSetGadgetHandler (FormType *formP, Uint16 objIndex, FormGadgetHandlerType *attrP)

Parameters
- `formP` Pointer to the form object (FormType structure).
- `objIndex` Index of a gadget object in the form. You can obtain this by using FrmGetObjectIndex.
- `attrP` Address of the callback function. See FormGadgetHandlerType.

Result
Returns nothing.

Comments
This function sets the application-defined function that controls the specified gadget’s behavior. This function is called when the gadget needs to be drawn, erased, deleted, or needs to handle an event.

Compatibility
Implemented only if 3.5 New Feature Set is present.

See Also
FrmGetGadgetData, FrmSetGadgetData
**FrmSetMenu**

**Purpose**  
Change a form’s menu bar and make the new menu active.

**Declared In**  
Form.h

**Prototype**  
void FrmSetMenu (FormType *formP, UInt16 menuRscID)

**Parameters**
- `formP`  
  Pointer to the form object (FormType structure).
- `menuRscID`  
  Resource ID of the menu.

**Result**  
Returns nothing.

**Compatibility**  
Implemented only if 2.0 New Feature Set is present.

**FrmSetObjectBounds**

**Purpose**  
Set the bounds or position of an object.

**Declared In**  
Form.h

**Prototype**  
void FrmSetObjectBounds (FormType *formP, UInt16 objIndex, const RectangleType *bounds)

**Parameters**
- `formP`  
  Pointer to the form object (FormType structure).
- `objIndex`  
  Index of an object in the form. You can obtain this by using FrmGetObjectIndex.
- `bounds`  
  Window-relative bounds. For the following objects, this sets only the position of the top-left corner: label, bitmap, and Graffiti state indicator.

**Result**  
Returns nothing.
Comments  Doesn’t update the display.

Compatibility  Implemented only if 2.0 New Feature Set is present.

**FrmSetObjectPosition**

**Purpose**  Set the position of an object.

**Declared In**  Form.h

**Prototype**  

```c
void FrmSetObjectPosition (FormType *formP, 
UInt16 objIndex, Coord x, Coord y)
```

**Parameters**

- `-> formP`  Pointer to the form object (FormType structure).
- `-> objIndex`  Index of an object in the form. You can obtain this by using FrmGetObjectIndex.
- `-> x`  Window-relative horizontal coordinate.
- `-> y`  Window-relative vertical coordinate.

**Result**  Returns nothing.

**See Also**  FrmGetObjectPosition, FrmGetObjectBounds

**FrmSetTitle**

**Purpose**  Set the title of a form. If the form is visible, draw the new title.

**Declared In**  Form.h

**Prototype**  

```c
void FrmSetTitle (FormType *formP, 
Char *newTitle)
```

**Parameters**

- `-> formP`  Pointer to the form object (FormType structure).
-> newTitle Pointer to the new title string.

**Result**
Returns nothing.

**Comments**
This function draws the title if the form is visible.

This function saves the pointer passed in newTitle; it does **not** make a copy. The value of newTitle must not be a pointer to a stack-based object.

**Compatibility**
Earlier versions of this function redrew the title without first erasing the old one. This problem was corrected in version 3.0 of Palm OS.

**See Also**
FrmGetTitle, FrmCopyTitle, FrmCopyLabel

---

## FrmShowObject

**Purpose**
Set a form object as usable. If the form is visible, draw the object.

**Declared In**
Form.h

**Prototype**

```
void FrmShowObject (FormType *formP,
                   UInt16 objIndex)
```

**Parameters**

- `-> formP` Pointer to the form object (**FormType** structure).

- `-> objIndex` Index of an object in the form. You can obtain this by using **FrmGetObjectIndex**.

**Result**
Returns nothing.

**Compatibility**
On versions of Palm OS prior to 3.5 this function doesn’t affect lists or tables. On Palm OS 3.5 it operates correctly on lists but doesn’t have any effect on tables. On Palm OS 4.0 it operates correctly on both lists and tables.
If 3.5 New Feature Set is present and the object is an extended gadget, this function calls the gadget’s callback with formGadgetDrawCmd. See FormGadgetHandlerType.

See Also  FrmHideObject

FrmUpdateForm

Purpose  Send a frmUpdateEvent to the specified form.

Declared In  Form.h

Prototype  void FrmUpdateForm (UInt16 formId, UInt16 updateCode)

Parameters  

- > formId  Resource ID of form to update.

- > updateCode  An application-defined code that can be used to indicate what needs to be updated. Specify the code frmRedrawUpdateCode to indicate that the whole form should be redrawn.

Result  Returns nothing.

Comments  If the frmUpdateEvent posted by this function is handled by the default form event handler, FrmHandleEvent, the updateCode parameter is ignored. FrmHandleEvent always redraws the form.

If you handle the frmUpdateEvent in a custom event handler, you can use the updateCode parameter any way you want. For example, you might use it to indicate that only a certain part of the form needs to be redrawn. If you do handle the frmUpdateEvent, be sure to return true from your event handler so that the default form handler does not also redraw the whole form.

If you do handle the frmUpdateEvent in a custom event handler, be sure to handle the case where updateCode is set to frmRedrawUpdateCode, and redraw the whole form. This event (and code) is sent by the system when the whole form needs to be redrawn because the display needs to be refreshed.
**FrmUpdateScrollers**

**Purpose**
Visually update (show or hide) the field scroll arrow buttons.

**Declared In**
Form.h

**Prototype**
```c
void FrmUpdateScrollers (FormType *formP,
UInt16 upIndex, UInt16 downIndex,
Boolean scrollableUp, Boolean scrollableDown)
```

**Parameters**
- `formP` Pointer to the form object (*FormType* structure).
- `upIndex` Index of the up-scroller button. You can obtain this by using `FrmGetObjectIndex`.
- `downIndex` Index of the down-scroller button. You can obtain this by using `FrmGetObjectIndex`.
- `scrollableUp` Set to *true* to make the up scroll arrow active (shown), or *false* to hide it.
- `scrollableDown` Set to *true* to make the down scroll arrow active (shown), or *false* to hide it.

**Result**
Returns nothing.

**FrmValidatePtr**

**Purpose**
Return *true* if the specified pointer references a valid form.

**Declared In**
Form.h

**Prototype**
```c
Boolean FrmValidatePtr (const FormType *formP)
```

**Parameters**
- `formP` Pointer to be tested.

**Result**
Returns *true* if the specified pointer is a non-NULL pointer to an object having a valid form structure.
Comments  This function is intended for debugging purposes only. Do not include it in released code.
To distinguish between a window and a form in released code, instead of using this function, look at the flag `windowFlags.dialog` in the `WindowType` structure. This flag is true if the window is a form.

**Compatibility**
Implemented only if [3.0 New Feature Set](#) is present.

### FrmVisible

**Purpose**
Return true if the form is visible (is drawn).

**Declared In**
Form.h

**Prototype**
```c
Boolean FrmVisible (const FormType *formP)
```

**Parameters**
- `formP` Pointer to the form object (`FormType` structure).

**Result**
Returns true if the form is visible; false if it is not visible.

**See Also**
`FrmDrawForm`, `FrmEraseForm`

### Application-Defined Functions

#### FormCheckResponseFuncType

**Purpose**
Callback function for `FrmCustomResponseAlert`.

**Declared In**
Form.h

**Prototype**
```c
Boolean FormCheckResponseFuncType (Int16 button, Char *attempt)
```

**Parameters**
- `button` The ID of the button that the user tapped.
-> attempt The string that the user entered in the alert dialog.

**Result**
Return **true** if the dialog should be dismissed. Return **false** if the dialog should not be dismissed.

**Comments**
This function is called at these times during the FrmCustomResponseAlert routine:

- At the beginning of FrmCustomResponseAlert, this function is called with a button ID of frmResponseCreate. This constant indicates that the dialog is about to be displayed, and your function should perform any necessary initialization. For example, on a Japanese system, a password dialog might need to disable the Japanese FEP. So it would call TsmSetFepMode(NULL, tsmFepModeOff) in this function.

- When the user has tapped a button on the dialog. The function should process the attempt string. If the string is valid input, the function should return **true**. If not, it should return **false** to give the user a chance to re-enter the string.

- At the end of FrmCustomResponseAlert, this function is called with a button ID of frmResponseQuit. This gives the callback a chance to perform any cleanup, such as re-enabling the Japanese FEP.

**Compatibility**
Implemented only if 3.5 New Feature Set is present.
### FormEventHandlerType

**Purpose**
The event handler callback routine for a form.

**Declared In**
Form.h

**Prototype**
Boolean FormEventHandlerType (EventType *eventP)

**Parameters**
- `eventP` Pointer to the form event (FormType structure).

**Result**
Must return `true` if this routine handled the event, otherwise `false`.

**Comments**
FrmDispatchEvent calls this handler whenever it receives an event for the form.

This callback routine is the mechanism for dispatching events to particular forms in an application. The callback is registered by the routine FrmSetEventHandler.

### FormGadgetHandlerType

**Purpose**
The event handler callback for an extended gadget.

**Declared In**
Form.h

**Prototype**
Boolean (FormGadgetHandlerType)
(struct FormGadgetTypeInCallback *gadgetP, Uint16 cmd, void *paramP)

**Parameters**
- `gadgetP` Pointer to the gadget structure. See FormGadgetType.
- `cmd` A constant that specifies what action the handler should take. This can be one of the following:
  - formGadgetDeleteCmd
    Sent by FrmDeleteForm to indicate that
the gadget is being deleted and must clean up any memory it has allocated or perform other cleanup tasks.

**formGadgetDrawCmd**

Sent by **FrmDrawForm** and **FrmShowObject** to indicate that the gadget must be drawn or redrawn.

**formGadgetEraseCmd**

Sent by **FrmHideObject** to indicate that the gadget is going to be erased. FrmHideObject clears the visible and usable flags for you. If you return false, it also calls WinEraseRectangle to erase the gadget’s bounds.

**formGadgetHandleEventCmd**

Sent by **FrmHandleEvent** to indicate that a gadget event has been received. The paramp parameter contains the pointer to the EventType structure.

-> paramP  NULL except if cmd is formGadgetHandleEventCmd. In that case, this parameter holds the pointer to the EventType structure containing the event.

**Result**  Return true if the event was handed successfully; false otherwise.

**Comments**  If this function performs any drawing in response to the formGadgetDrawCmd, it should set the gadget’s visible attribute flag. (gadgetP->attr.visible = true). This flag indicates that the gadget appears on the screen. If you don’t set the visible flag, the gadget won’t be erased when FrmHideObject is called. (FrmHideObject immediately returns if the object’s visible flag is false.)

Note that if the function receives the formGadgetEraseCmd, it may simply choose to perform any necessary cleanup and return false. If the function returns false, FrmHideObject erases the
gadget’s bounding rectangle. If the function returns true, it must erase the gadget area itself.

If this function receives a `formGadgetHandleEventCmd`, `paramP` points one of two events: `frmGadgetEnterEvent` or `frmGadgetMiscEvent`. The `frmGadgetEnterEvent` is passed when there is a `penDownEvent` within the gadget’s bounds. This function should track the pen and perform any necessary highlighting. The `frmGadgetMiscEvent` is never sent by the system. Your application may choose to use it if at any point it needs to send data to the extended gadget. In this case, the event has one or both of these fields defined: `selector`, an unsigned integer, and `dataP`, a pointer to data.

**Compatibility**

Implemented only if [3.5 New Feature Set](#) is present.

**See Also**

FrmSetGadgetHandler
Forms
Application-Defined Functions
Graffiti Shift

This chapter provides reference material for the Graffiti® Shift facility, declared in the header file GraffitiShift.h.

GraffitiShift Functions

GsiEnable

Purpose
Enable or disable the Graffiti-shift state indicator.

Declared In
GraffitiShift.h

Prototype
void GsiEnable (const Boolean enableIt)

Parameters
enableIt true to enable, false to disable.

Result
Returns nothing.

Comments
Enabling the indicator makes it visible, disabling it makes the insertion point invisible.
Graffiti Shift

GraffitiShift Functions

GsiEnabled

Purpose  Return true if the Graffiti-shift state indicator is enabled, or false if it's disabled.

Declared In  GraffitiShift.h

Prototype  Boolean GsiEnabled (void)

Parameters  None.

Result  true if enabled, false if not.

GsiInitialize

Purpose  Initialize the global variables used to manage the Graffiti-shift state indicator.

Declared In  GraffitiShift.h

Prototype  void GsiInitialize (void)

Parameters  None.

Result  Returns nothing.

GsiSetLocation

Purpose  Set the display-relative position of the Graffiti-shift state indicator.

Declared In  GraffitiShift.h

Prototype  void GsiSetLocation (const Int16 x, const Int16 y)

Parameters  x, y  Coordinate of left side and top of the indicator.

Result  Returns nothing.
Graffiti Shift
GraffitiShift Functions

Comments
The indicator is not redrawn by this routine.

GsiSetShiftState

Purpose
Set the Graffiti-shift state indicator.

Declared In
GraffitiShift.h

Prototype
void GsiSetShiftState (const UInt16 lockFlags,
const UInt16 tempShift)

Parameters
lockFlags    glfCapsLock or glfNumLock.
tempShift    The current temporary shift.

Result
Returns nothing.

Comments
This function affects only the state of the UI element, not the underlying Graffiti engine.

See Also
GrfSetState
Insertion Point

This chapter provides reference material for the insertion point API, declared in the header file \texttt{InsPoint.h}.

For more information on the insertion point, see the section “Insertion Point” in the \textit{Palm OS Programmer’s Companion}, vol. I.

Insertion Point Functions

\textbf{InsPtEnable}

\begin{description}
\item[Purpose] Enable or disable the insertion point. When the insertion point is disabled, it’s invisible; when it’s enabled, it blinks.
\item[Declared In] \texttt{InsPoint.h}
\item[Prototype] \texttt{void InsPtEnable (Boolean enableIt)}
\item[Parameters] \texttt{enableIt} \hspace{1em} true = enable; false = disable
\item[Result] Returns nothing.
\item[Comments] This function is called by the Form functions when a text field loses or gains the focus, and by the Windows function when a region of the display is copied (\texttt{WinCopyRectangle}).
\item[See Also] \texttt{InsPtEnabled}
\end{description}
InsPtEnabled

Purpose  Return true if the insertion point is enabled or false if the insertion point is disabled.

Declared In  InsPoint.h

Prototype  Boolean InsPtEnabled (void)

Parameters  None.

Result  Returns true if the insertion point is enabled (blinking); returns false if the insertion point is disabled (invisible).

See Also  InsPtEnable

InsPtGetHeight

Purpose  Return the height of the insertion point.

Declared In  InsPoint.h

Prototype  Int16 InsPtGetHeight (void)

Parameters  None.

Result  Returns the height of the insertion point, in pixels.
**InsPtGetLocation**

**Purpose**  
Return the screen-relative position of the insertion point.

**Declared In**  
InsPoint.h

**Prototype**  
void InsPtGetLocation (Int16 *x, Int16 *y)

**Parameters**  
 x, y  
Pointer to top-left position of insertion point’s x and y coordinate.

**Result**  
Returns nothing. Stores the location in x and y.

**Comments**  
This function is called by the Field functions. An application would not normally call this function.

**InsPtSetHeight**

**Purpose**  
Set the height of the insertion point.

**Declared In**  
InsPoint.h

**Prototype**  
void InsPtSetHeight (const Int16 height)

**Parameters**  
height  
Height of the insertion point in pixels.

**Result**  
Returns nothing.

**Comments**  
Set the height of the insertion point to match the character height of the font used in the field that the insertion point is in. When the current font is changed, the insertion point height should be set to the line height of the new font.

If the insertion point is visible when its height is changed, it’s erased and redrawn with its new height.

**See Also**  
InsPtGetHeight
**InsPtSetLocation**

**Purpose**
Set the screen-relative position of the insertion point.

**Declared In**
InsPoint.h

**Prototype**
void InsPtSetLocation (const Int16 x, const Int16 y)

**Parameters**
- x, y
  Number of pixels from the left side (top) of the display.

**Result**
Returns nothing.

**Comments**
The position passed to this function is the location of the top-left corner of the insertion point.
This function should be called only by the Field functions.

**See Also**
InsPtGetLocation
This chapter provides information about list objects by discussing these topics:

- **List Data Structures**
- **List Resources**
- **List Functions**
- **Application-Defined Function**

The header file `List.h` declares the API that this chapter describes. For more information on lists, see the section “Lists” in the *Palm OS Programmer’s Companion*, vol. I.

### List Data Structures

#### ListAttrType

The `ListAttrType` bit field defines the visible characteristics of the list.

```c
typedef struct {
    UInt16 usable       :1;
    UInt16 enabled      :1;
    UInt16 visible      :1;
    UInt16 poppedUp     :1;
    UInt16 hasScrollBar :1;
    UInt16 search       :1;
    UInt16 reserved      :2;  
} ListAttrType;
```
Field Descriptions

usable  Set to make the list usable.

If not set, the list is not considered part of the current interface of the application, and does not appear on screen.

enabled  Not used.

visible  Set when the list object is drawn, and cleared when the list object is erased.

This attribute is set and cleared internally.

poppedUp  Set to indicate that the choices are displayed in a popup window.

This attribute is set and cleared internally.

hasScrollBar  Set to indicate that the list has a scroll bar.

search  Set to enable incremental search. If incremental search is enabled, when the list is displayed the user can navigate the list by entering up to five characters. The list will scroll to present the first list item that matches the entered characters. This feature only works for popup lists, and only works if the list is sorted and the list items are available to the List Manager (that is, you don’t pass NULL to `LstSetListChoices`).

reserved  Reserved for system use.

ListType

The `ListType` structure is defined below.

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `ListType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.
typedef struct {
    UInt16        id;
    RectangleType bounds;
    ListAttrType  attr;
    Char          **itemsText;
    Int16         numItems;
    Int16         currentItem;
    Int16         topItem;
    FontID        font;
    UInt8         reserved;
    WinHandle     popupWin;
    ListDrawDataFuncPtr drawItemCallback;
} ListType;

Field Descriptions

id
The ID value, specified by the application
developer. This ID value is part of the event
data of lstEnterEvent and
lstSelectEvent.

bounds
The bounds of the list, relative to the
window. For example, to access the bounds
of an object in a form whose ID is kObjectID:
{
    RectangleType rect;
    FormPtr formP =
    FrmGetActiveForm();

    FrmGetObjectBounds(formP,
    FrmGetObjectIndex(formP,
    kObjectID),
    &rect);
}

attr
The list attributes. See ListAttrType.
A pointer to an array of pointers to the text of the choices. Access with \texttt{LstGetSelectionText}. For example, to access the string specified by \texttt{itemNum} in the list whose ID is \texttt{kChoiceList} use the following:

\begin{verbatim}
{
  Char *string;
  Int16 itemNum;
  ...
  string = LstGetSelectionText(GetObjectPtr(kChoicesList), itemNum);
}
\end{verbatim}

where \texttt{GetObjectPtr} is the following:

\begin{verbatim}
static void *GetObjectPtr(UInt16 rsrcID){
  FormPtr formP;
  formP = FrmGetActiveForm();
  return FrmGetObjectPtr(formP, FrmGetObjectIndex(formP, rsrcID));
}
\end{verbatim}

If you use a callback routine to draw the list items, note that the \texttt{itemsText} pointer you supply to \texttt{LstSetListChoices} is passed to your callback routine. See the comments under \texttt{ListDrawDataFuncType} for tips on using \texttt{itemsText} with a callback routine.

The number of choices in the list. Access with \texttt{LstGetNumberOfItems}.

The currently-selected list choice (0 = first choice). Access with \texttt{LstGetSelection}.

The first choice displayed in the list. Set this field with \texttt{LstSetTopItem}. 

Lists
List Resources

List Resources
The List Resource (tLST), and Popup Trigger Resource (tPUT) are used together to represent an active list.

List Functions

LstDrawList

Purpose Sets the visible attribute of the list object to true, and draws the list object if it is usable.

Declared In List.h

Prototype void LstDrawList (ListType *listP)

Parameters -> listP Pointer to a list object (ListType).

Result Returns nothing.

Comments If there are more choices than can be displayed, this function ensures that the current selection is visible. The current selection is highlighted. Note that this function does not ensure the current
selection is visible; if you need to do this, call the
LstMakeItemVisible function.

If the list is disabled, it’s drawn grayed-out (strongly discouraged).
If it’s empty, nothing is drawn. If it’s not usable, nothing is drawn.

See Also  FrmGetObjectPtr, LstPopupList, LstEraseList

LstEraseList

Purpose   Erase a list object.

Declared In   List.h

Prototype   void LstEraseList (ListType *listP)

Parameters   -> listP       Pointer to a list object (ListType).

Result   Returns nothing.

Comments   The visible attribute is set to false by this function.

See Also  FrmGetObjectPtr, LstDrawList

LstGetNumberOfItems

Purpose   Return the number of items in a list.

Declared In   List.h

Prototype   Int16 LstGetNumberOfItems (const ListType *listP)

Parameters   -> listP       Pointer to a list object (ListType).

Result   Returns the number of items in a list.

See Also  FrmGetObjectPtr, LstSetListChoices
**LstGetSelection**

**Purpose**
Return the currently selected choice in the list.

**Declared In**
List.h

**Prototype**

```c
Int16 LstGetSelection (const ListType *listP)
```

**Parameters**
- `listP` Pointer to a list object.

**Result**
Returns the item number of the current list choice. The list choices are numbered sequentially, starting with 0; Returns `noListSelection` if none of the items are selected.

**See Also**
FrmGetObjectPtr, LstSetListChoices, LstSetSelection, LstGetSelectionText

**LstGetSelectionText**

**Purpose**
Return a pointer to the text of the specified item in the list, or NULL if no such item exists.

**Declared In**
List.h

**Prototype**

```c
Char *LstGetSelectionText (const ListType *listP, Int16 itemNum)
```

**Parameters**
- `listP` Pointer to a list object.
- `itemNum` Item to select (0 = first item in list).

**Result**
Returns a pointer to the text of the current selection, or NULL if out of bounds.

**Comments**
This is a pointer within ListType, not a copy. This function is only usable if you supplied an array of strings and a count to
See Also **FrmGetObjectPtr**

**LstGetTopItem**

**Purpose**
Returns the topmost visible item.

**Declared In**
List.h

**Prototype**

```
Int16 LstGetTopItem (const ListType *listP)
```

**Parameters**
- `listP` Pointer to a list object.

**Result**
Returns the item number of the top item visible.

**Compatibility**
Implemented only if [4.0 New Feature Set](#) is present.

**See Also**
LstGetSelection, LstSetTopItem

**LstGetVisibleItems**

**Purpose**
Return the number of visible items.

**Declared In**
List.h

**Prototype**

```
Int16 LstGetVisibleItems (const ListType *listP)
```

**Parameters**
- `listP` Pointer to a list object.

**Result**
The number of items visible.

**Compatibility**
Implemented only if [2.0 New Feature Set](#) is present.
**LstHandleEvent**

**Purpose**  Handle event in the specified list; the list object must have its usable and visible attribute set to true. This routine handles two type of events, `penDownEvent` and `lstEnterEvent`; see Comments.

**Declared In**  List.h

**Prototype**  
```c
Boolean LstHandleEvent (ListType *listP,
const EventType *eventP)
```

**Parameters**  
- `listP`  Pointer to a list object (ListType).
- `eventP`  Pointer to an EventType structure.

**Result**  Return true if the event was handled. The following cases will result in a return value of true:
- A `penDownEvent` within the bounds of the list
- A `lstEnterEvent` with a list ID value that matches the list ID in the list data structure

**Comments**  When this routine receives a `penDownEvent`, it checks if the pen position is within the bounds of the list object. If it is, this routine tracks the pen until the pen comes up. If the pen comes up within the bounds of the list, a `lstEnterEvent` is added to the event queue, and the routine is exited.

When this routine receives a `lstEnterEvent`, it checks that the list ID in the event record matches the ID of the specified list. If there is a match, this routine creates and displays a popup window containing the list’s choices and the routine is exited.

If a `penDownEvent` is received while the list’s popup window is displayed and the pen position is outside the bounds of the popup window, the window is dismissed. If the pen position is within the bounds of the window, this routine tracks the pen until it comes up. If the pen comes up outside the list object, a `lstEnterEvent` is added to the event queue.
**LstMakeItemVisible**

**Purpose**
Make an item visible, preferably at the top. If the item is already visible, make no changes.

**Declared In**
List.h

**Prototype**
```c
void LstMakeItemVisible (ListType *listP, Int16 itemNum)
```

**Parameters**
- `listP` Pointer to a list object (`ListType`).
- `itemNum` Item to select (0 = first item in list).

**Result**
Returns nothing.

**Comments**
Does not visually update the list. You must call `LstDrawList` to update it.

**See Also**
`FrmGetObjectPtr`, `LstSetSelection`, `LstSetTopItem`, `LstDrawList`
**LstNewList**

**Purpose**  Create a new list object dynamically and install it in the specified form. This function can be used to create a new popup trigger and its associated list.

**Declared In**  List.h

**Prototype**  
```c
Err LstNewList (void **formPP, UInt16 id,
Coord x, Coord y, Coord width, Coord height,
FontID font, Int16 visibleItems, Int16 triggerId)
```

**Parameters**

<--> **formPP**  
Pointer to the pointer to the form in which the new list is installed. This value is not a handle; that is, the old `formPP` value is not necessarily valid after this function returns. In subsequent calls, always use the new `formPP` value returned by this function.

-> **id**  
Symbolic ID of the list, specified by the developer. By convention, this ID should match the resource ID (not mandatory).

-> **x**  
Horizontal coordinate of the upper-left corner of the list’s boundaries, relative to the window in which it appears.

-> **y**  
Vertical coordinate of the upper-left corner of the list’s boundaries, relative to the window in which it appears.

-> **width**  
Width of the list, expressed in pixels. Valid values are 1 – 160.

-> **height**  
Height of the list, expressed in pixels. Valid values are 1 – 160.

-> **visibleItems**  
Number of list items that can be viewed together.
Lists
List Functions

-> triggerId  Symbolic ID of the popup trigger associated with the new list (this ID is specified by the developer). A nonzero value for triggerId causes this function to create both the list and its associated popup trigger. If the list isn’t a popup list, pass 0 for triggerId.

Result  Returns 0 if no error.

Compatibility  Implemented only if 3.0 New Feature Set is present.

See Also  LstDrawList, FrmRemoveObject

LstPopupList

Purpose  Display a modal window that contains the items in the list.

Declared In  List.h

Prototype  Int16 LstPopupList (ListType *listP)

Parameters  -> listP  Pointer to a list object.

Result  Returns the list item selected, or -1 if no item was selected.

Comments  Saves the previously active window. Creates and deletes the new popup window.

See Also  FrmGetObjectPtr
**LstScrollList**

**Purpose** Scroll the list up or down a number of times.

**Declared In** List.h

**Prototype**

```c
Boolean LstScrollList (ListType *listP, WinDirectionType direction, Int16 itemCount)
```

**Parameters**

- `listP` Pointer to a list object.
- `direction` Direction to scroll.
- `itemCount` Items to scroll in direction.

**Result** Returns `true` when the list is actually scrolled, `false` otherwise. May return `false` if a scroll past the end of the list is requested.

**Compatibility** Implemented only if 2.0 New Feature Set is present.

**LstSetDrawFunction**

**Purpose** Set a callback function to draw each item instead of drawing the item’s text string.

**Declared In** List.h

**Prototype**

```c
void LstSetDrawFunction (ListType *listP, ListDrawDataFuncPtr func)
```

**Parameters**

- `listP` Pointer to a list object.
- `func` Pointer to a function that draws items.

**Result** Returns nothing.
Comments This function also adjusts topItem to prevent a shrunken list from being scrolled down too far. Use this function for custom draw functionality.

See Also FrmGetObjectPtr, LstSetListChoices, ListDrawDataFuncType

LstSetHeight

Purpose Set the number of items visible in a list.

Declared In List.h

Prototype void LstSetHeight (ListType *listP, Int16 visibleItems)

Parameters -> listP Pointer to a list object.
-> visibleItems Number of choices visible at once.

Result Returns nothing.

Comments This function doesn’t redraw the list if it’s already visible.

See Also FrmGetObjectPtr

LstSetListChoices

Purpose Set the items of a list to the array of text string pointers passed to this function. This functions erases the old list items.

Declared In List.h

Prototype void LstSetListChoices (ListType *listP, Char **itemsText, Int16 numItems)

Parameters -> listP Pointer to a list object.
Lists
List Functions

-> itemsText  Pointer to an array of text strings. See SysFormPointerArrayToStrings for one way to create this array of strings.

-> numItems  Number of choices in the list.

Result  Returns nothing.

Comments  You need to call the LstDrawList function to update the list if it is displayed when you call this function.

NOTE:  This function does not copy the strings in the itemsText array, which means that you need to ensure that the array is not moved or deallocated until after you are done with the list.

If you use a callback routine to draw the items in your list, the itemsText pointer is simply passed to that callback routine and is not otherwise used by the List Manager code. See the comments under ListDrawDataFuncType for tips on using the itemsText parameter with a callback routine.

See Also  FrmGetObjectPtr, LstSetSelection, LstSetTopItem, LstDrawList, LstSetHeight, LstSetDrawFunction

LstSetPosition

Purpose  Set the position of a list.

Declared In  List.h

Prototype  void LstSetPosition (ListType *listP, Coord x, Coord y)

Parameters  -> listP  Pointer to a list object
             -> x  Left bound.


Lists
List Functions

- > y Top bound.

Result
Returns nothing.

Comments
The list is not redrawn. Don’t call this function when the list is visible.

See Also
FrmGetObjectPtr

LstSetSelection

Purpose
Set the selection for a list.

Declared In
List.h

Prototype
void LstSetSelection (ListType *listP, Int16 itemNum)

Parameters
- > listP Pointer to a list object.
- > itemNum Item to select (0 = first item in list, noListSelection = none).

Result
Returns nothing.

Comments
The old selection, if any, is unselected. If the list is visible, the selected item is visually updated. The list is scrolled to the selection, if necessary, as long as the list object is both visible and usable.

See Also
FrmGetObjectPtr, LstSetTopItem
LstSetTopItem

Purpose
Set the item visible. The item cannot become the top item if it’s on
the last page.

Declared In
List.h

Prototype
void LstSetTopItem (ListType *listP,
Int16 itemNum)

Parameters
-> listP Pointer to a list object.
-> itemNum Item to select (0 = first item in list). This must be
a valid item number.

Result
Returns nothing.

Comments
Does not update the display.

NOTE: The value you specify for itemNum must be in the range
0 to max-number-of-items.

See Also
FrmGetObjectPtr, LstSetSelection, LstGetTopItem,
LstMakeItemVisible, LstDrawList, LstEraseList

Application-Defined Function

If you need to perform special drawing for items in the list, call
LstSetDrawFunction to set the list drawing callback function.
The ListDrawDataFuncType section documents the prototype
for the callback function you provide for drawing list items.

ListDrawDataFuncType

Purpose
Callback function that you provide for drawing items in a list. This
function is called whenever the Palm OS needs to draw an element
Lists
Application-Defined Function

in the list. Your callback function declaration must match the
prototype shown here.

Declared In List.h

Prototype void ListDrawDataFuncType(Int16 itemNum,
RectangleType *bounds, Char **itemsText)

Parameters -> itemNum The number of the item to draw.
-> bounds The bounds of the list, relative to the window.
-> itemsText A pointer to an array of pointers to the text of
the list items. This is the pointer that you
supplied when calling LstSetListChoices.

Result Returns nothing.

Comments You can call LstSetDrawFunction to register your callback
function for the list, which means that your function will be called
to draw the list items, rather than using the built-in draw
functionality, which displays each item’s text string.

Your callback function is called whenever an item in the list needs to
be drawn. When it is called, the value of the itemNum parameter
specifies which item the function is to draw. The itemsText
parameter, which is what was supplied to LstSetListChoices,
doesn’t actually need to point to a list of strings: you can pass NULL,
or you can pass a pointer to anything that will be useful to your
drawing function. Note, however, that if you pass anything other
than a pointer to a list of strings when you call
LstSetListChoices, you must ensure that
LstGetSelectionText is never called since it assumes that this
pointer indicates an array of text items. In particular, if your list is
associated with a pop-up trigger you must handle the
popSelectEvent yourself before FrmHandleEvent gets a chance
at it since FrmHandleEvent calls LstGetSelectionText.
**WARNING!** If the list is a popup list, the form that owns the list is not active while the list is in a window. This means that you cannot call the `FrmGetActiveForm` function. Instead, use `itemsText` pointer to access any information that you need for drawing. If you must access the form, use the `FrmGetFormPtr` function.

Note that the list object manages which colors are used to draw the items and how to draw selected versus unselected items. In almost all circumstances, your drawing function does not have to be concerned with these details.

However, if you do need to determine if the item is selected, you can rely on the fact that the system has set the pen color to one of two colors prior to calling your draw function:

- If the item is selected, the foreground color is `UIObjectSelectedForeground`.
- If the item is not selected, the foreground color is `UIObjectForeground`.

You can determine the foreground color that is in effect for the list item by calling the `WinSetForeColor` function, which returns the previous value of the foreground color. Remember to call `WinSetForeColor` again to reset the foreground color to what it was. For example:

```plaintext
itemColor = WinSetForeColor(myColor)
WinSetForeColor(itemColor)
selectColor = UiColorGetTableEntryIndex(UIObjectSelectedForeground)
if itemColor == selectColor
...
```

**See Also**  
`LstSetDrawFunction`, `UIColorGetTableEntryIndex`, `WinSetForeColor`
Lists
Application-Defined Function
Menus

This chapter describes the menu API as declared in the header file Menu.h. It discusses the following topics:

- **Menu Data Structures**
- **Menu Constants**
- **Menu Resources**
- **Menu Functions**

For more information on menus, see the section “Menus” on page 105 in the *Palm OS Programmer’s Companion*, vol. I.

**Menu Data Structures**

**MenuBarAttrType**

The `MenuBarAttrType` bit field defines some characteristics of the menu bar.

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `MenuBarAttrType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```c
typedef struct {
    UInt16 visible : 1;
    UInt16 commandPending : 1;
    UInt16 insPtEnabled : 1;
    UInt16 needsRecalc : 1;
} MenuBarAttrType;
```
Menus
Menu Data Structures

Your code should treat the MenuBarAttrType structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

**Field Descriptions**

- **visible**: If set, the menu bar is drawn and visible on the screen. This attribute is set as part of MenuDrawMenu, which is called when the menu is drawn.

- **commandPending**: If set, a menu command shortcut is in progress. This bit is set during MenuHandleEvent if the menu shortcut keystroke is received. If the next key is received before the timeout value is reached, the key is examined to see if it is a valid menu command.

- **insPtEnabled**: Stores the state of the insertion point at the time the menu was drawn so that it can be restored when the menu is erased.

- **needsRecalc**: If set, recalculate menu dimensions.

**Compatibility**

The needsRecalc constant is present only if 3.5 New Feature Set is present.

**MenuCmdBarButtonType**

The MenuCmdBarButtonType structure defines a button to be displayed on the command toolbar. The buttonsData field of the MenuCmdBarType structure contains an array of structures of this type.
WARNING! PalmSource, Inc. does not support or provide backward compatibility for the MenuCmdBarButtonType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```c
typedef struct {
    UInt16   bitmapId;
    Char     name[menuCmdBarMaxTextLength];
    MenuCmdBarResultType resultType;
    UInt8    reserved;
    UInt32   result;
} MenuCmdBarButtonType;
```

Your code should treat the MenuCmdBarButtonType structure as opaque. Do not attempt to change structure member values directly. Instead, use MenuCmdBarAddButton to add a button to the display. For the most part, the parameters to MenuCmdBarAddButton are the same as the fields in the MenuCmdBarButtonType, so there should be no need to alter these fields directly.

MenuCmdBarGetButtonData can be called to access information about command bar buttons.

**Field Descriptions**

- **bitmapId**: Resource ID of the bitmap to display on the button. This bitmap should be 13 pixels high by 16 pixels wide.
- **name**: Text to display in the status message when the user taps the button.
- **resultType**: Specifies what type of data is contained in the result field. See MenuCmdBarResultType.
Menus
Menu Data Structures

reserved  Reserved for future use.
result  Specifies the data to send when the user clicks the button. The data is interpreted as specified by the resultType field. The result can be a shortcut character to enqueue in a keyDownEvent, a menu item ID to enqueue in a menuEvent, or a notification to be broadcast.

Compatibility  This structure is defined only if 3.5 New Feature Set is present.

MenuCmdBarResultType

The MenuCmdBarResultType enum specifies how the result field in the MenuCmdBarButtonType structure should be interpreted.

typedef enum {
    menuCmdBarResultNone,
    menuCmdBarResultChar,
    menuCmdBarResultMenuItem,
    menuCmdBarResultNotify
} MenuCmdBarResultType;

Value Descriptions

menuCmdBarResultNone  Send nothing.
menuCmdBarResultChar  The result is a character to send in a keyDownEvent.
menuCmdBarResultMenuItem  The result is the ID of the menu item to send in a menuEvent.
menuCmdBarResultNotify  The result is a notification constant to be broadcast using SysNotifyBroadcastDeferred

Compatibility  This enum is defined only if 3.5 New Feature Set is present.
MenuCmdBarType

The MenuCmdBarType structure defines the command toolbar. This command toolbar is allocated and displayed when the user draws the shortcut stroke in the Graffiti® area. It is deallocated when MenuEraseStatus is called, which occurs most frequently when the timeout value has elapsed.

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the MenuCmdBarType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```c
typedef struct MenuCmdBarType {
    WinHandle    bitsBehind;
    Int32        timeoutTick;
    Coord        top;
    Int16        numButtons;
    Boolean      insPtWasEnabled;
    Boolean      gsiWasEnabled;
    Boolean      feedbackMode;
    MenuCmdBarButtonType *buttonsData;
} MenuCmdBarType;
```

Your code should treat the MenuCmdBarType structure as opaque. Do not attempt to change structure member values directly.

**Field Descriptions**

- **bitsBehind**
  Handle for the window that contains the region obscured by the command toolbar.

- **timeoutTick**
  Timeout value given in system ticks. If the user hasn’t specified a command after this many ticks, the command toolbar is erased from the screen and deallocated from memory. This value also specifies how long the status message is displayed after the user successfully enters a command.
Menus

Menu Data Structures

(MenuData) MenuBarType

- **top**
  The top bound of the command toolbar given in display-relative coordinates. The command toolbar is as wide as the screen and displays at the bottom of the screen.

- **numButtons**
  Number of buttons displayed on the command toolbar.

- **insPtWasEnabled**
  If true, the insertion point was enabled before the command toolbar was displayed and should be re-enabled when the command toolbar is erased. If false, the insertion point was disabled.

- **gsiWasEnabled**
  If true, the Graffiti shift indicator was enabled before the command toolbar was displayed and should be re-enabled when the command toolbar is erased. If false, the Graffiti shift indicator was disabled.

- **feedbackMode**
  If true, the command toolbar is currently displaying a status message. The status message is displayed to tell the user what command is being performed. If false, the command toolbar is awaiting input.

- **buttonsData**
  The list of buttons to display on the command toolbar. See MenuCmdBarButtonType. Buttons are stored in this list sequentially with the rightmost button at index 0. Access with MenuCmdBarGetButtonData.

Compatibility

This structure is defined only if 3.5 New Feature Set is present.

**MenuBarPtr**

The MenuBarPtr type defines a pointer to a MenuBarType.
typedef MenuBarType *MenuBarPtr;

**MenuBarType**
The MenuBarType structure defines the menu bar. There is one menu bar per form.

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the MenuBarType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```c
typedef struct {
    WinHandle barWin;
    WinHandle bitsBehind;
    WinHandle savedActiveWin;
    WinHandle bitsBehindStatus;
    MenuBarAttrType attr;
    Int16 curMenu;
    Int16 curItem;
    Int32 commandTick;
    Int16 numMenus;
    MenuPullDownPtr menus;
} MenuBarType;
```

Your code should treat the MenuBarType structure as opaque. Do not attempt to change structure member values directly.

**Field Descriptions**

- **barWin**
  Handle for the window that contains the menu bar.

- **bitsBehind**
  Handle for the window that contains the region obscured by the menu bar.

- **savedActiveWin**
  Handle where the currently active window is saved so that it can be restored when the menu is erased.
**Menus**

**Menu Data Structures**

- **bitsBehindStatus**: Handle where the bits behind the status message are saved so that when the message display terminates, the bits can be restored.

  The status message is displayed when the user activates the menu through the use of the command keystroke.

- **attr**: Menu bar attributes. See [MenuBarAttrType](#).

- **curMenu**: Menu number for the currently visible menu. Menus are numbered sequentially starting with 0. The value is preserved when the menu bar is dismissed. A value of `noMenuSelection` indicates that there is no current pull-down menu.

- **curItem**: Item number of the currently highlighted menu item. The items in each menu are numbered sequentially, starting with zero.

  A value of `noMenuItemSelection` indicates that there is no current item selected.

- **commandTick**: Tick count at which the status message should be erased.

- **numMenus**: Number of pull-down menus on the menu bar.

- **menus**: Array of [MenuPullDownType](#) structures.

**Compatibility**

If 3.5 New Feature Set is present, the `bitsBehindStatus` and `commandTick` fields are defined but are not used. Instead, the `bitsBehind` and `timeoutTick` fields in [MenuCmdBarType](#) define the save-behind window and the timeout value for the command toolbar.
MenuItemType

The MenuItemType structure defines a specific item within a menu. The items array in the MenuPullDownType structure contains one MenuItemType structure for each menu item in the pull-down menu.

If 3.5 New Feature Set is present, you can add a menu item to a pull-down menu programmatically using MenuAddItem.

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the MenuItemType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.
typedef struct {
    Uint16   id;
    Char     command;
    Uint8    hidden: 1;
    Uint8    reserved: 7;
    Char     *itemStr;
} MenuItemType;

Field Descriptions

id          ID value you specified when you created the menu item. This ID value is included as part of the event data of a menuEvent.

command     Shortcut key. If you provide shortcuts, make sure that each shortcut is unique among all commands available at that time.

hidden      If true, the menu item is hidden. If false, it is displayed. You can set and clear this value using MenuHideItem and MenuShowItem.

reserved    Reserved for future use.

itemStr     Pointer to the text to display for this menu item, including the shortcut key. To include a shortcut key, begin the string with the item’s text, then type a tab character, and then the item’s shortcut key.

            To create a separator bar, create a one-character string containing the MenuSeparatorChar constant.

Compatibility The hidden and reserved fields are defined only if 3.5 New Feature Set is present.
**MenuPullDownPtr**

The `MenuPullDownPtr` type defines a pointer to a `MenuPullDownType`.

```c
typedef MenuPullDownType *MenuPullDownPtr;
```

**MenuPullDownType**

The `MenuPullDownType` structure defines a specific menu accessed from the menu bar. The menus array in the `MenuBarType` structure contains one `MenuPullDownType` structure for each pull-down menu associated with the menu bar.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `MenuPullDownType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```c
typedef struct {
    WinHandle menuWin;
    RectangleType bounds;
    WinHandle bitsBehind;
    RectangleType titleBounds;
    Char *title;
    UInt16 hidden : 1;
    UInt16 numItems : 15;
    MenuItemType *items;
} MenuPullDownType;
```

**Field Descriptions**

- **menuWin**: Handle for the window that contains the menu.
- **bounds**: Position and size, in pixels, of the pull-down menu.
- **bitsBehind**: Handle of a window that contains the region obscured by the menu.
Menus
Menu Constants

title The menu title (null-terminated string) displayed in the menu bar.
titleBounds Position and size, in pixels, of the title in the menu bar.
hidden If true, the menu is hidden; if false, it is displayed. This field is not currently used.
numItems Number of items in the menu. Separators count as items.
items Array of MenuItemType structures.

Compatibility The hidden field is defined only if 3.5 New Feature Set is present.

Menu Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noMenuSelection</td>
<td>-1</td>
<td>The curMenu field of MenuBarType is set to this when there is no currently selected menu.</td>
</tr>
<tr>
<td>noMenuItemSelection</td>
<td>-1</td>
<td>The curItem field of MenuBarType is set to this when there is no currently selected menu item.</td>
</tr>
<tr>
<td>separatorItemSelection</td>
<td>-2</td>
<td>The curItem field of MenuBarType is set to this when a menu separator item is selected.</td>
</tr>
<tr>
<td>MenuSeparatorChar</td>
<td>'–'</td>
<td>Special character indicating that the menu item is a bar used to separate groups of related menu items. The first character of the itemStr string in MenuItemType is set to this.</td>
</tr>
</tbody>
</table>
Menu Resources

The menu bar (MBAR) and pull-down menu (MENU) resources are used jointly to represent a menu object on screen. See “Menus and Menu Bars” in Chapter 2, “Palm OS Resources.”

Menu Functions

MenuAddItem

Purpose
Adds an item to the currently active menu.

Declared In
Menu.h

Prototype
Err MenuAddItem (UInt16 positionId, UInt16 id, Char cmd, const Char *textP)

Parameters
- positionId
  ID of an existing menu item. The new menu item is added after this menu item.
- id
  ID value to use for the new menu item.
- cmd
  Shortcut key. If you provide shortcuts, make sure that each shortcut is unique among all commands available at that time.
- textP
  Pointer to the text to display for this menu item, including the shortcut key. To include a shortcut key, begin the string with the item’s text, then type a tab character, and then the item’s shortcut key.
  To create a separator bar, create a one-character string containing the MenuSeparatorChar constant.

Result
Returns 0 upon success or one of the following if an error occurs:

- menuErrNoMenu
  The textP parameter is NULL.
**Menus**

*Menu Functions*

menuErrSameId The menu already contains an item with the ID id.

menuErrNotFound The menu doesn’t contain an item with the ID positionId.

May display a fatal error message if there is no current menu.

**Comments**

This function creates a new MenuItemType structure and adds it to the MenuBarType’s item list.

You should call this function only in response to a menuOpenEvent. This event is generated when the menu is first made active. In general, a form’s menu becomes active the first time a keyDownEvent with a vchrMenu or vchrCommand is generated, and it remains active until a new form (including a modal form or alert panel) is displayed or until FrmSetMenu is called to change the form’s menu. Palm OS® user interface guidelines discourage adding or hiding menu items at any time other than when the menu is first made active.

**Compatibility**

Implemented only if 3.5 New Feature Set is present.

**MenuCmdBarAddButton**

**Purpose**

Defines a button to be displayed on the command toolbar.

**Declared In** Menu.h

**Prototype**

Err MenuCmdBarAddButton (UInt8 where, Uint16 bitmapId, MenuCmdBarResultType resultType, Uint32 result, Char *nameP)

**Parameters**

- where Either menuCmdBarOnLeft to add the button to the left of the other buttons on the command toolbar, menuCmdBarOnRight to add it to the right of the other buttons, or a number indicating the exact position of the button. Button positions are numbered from right to left, and the rightmost position is number 1.
-> bitmapId  Resource ID of the bitmap to display on the button. The bitmap’s dimensions should be 13 pixels high by 16 pixels wide.

-> resultType  The type of data contained in the result parameter. See **MenuCmdBarResultType**.

-> result  The data to send when the user taps this button. This can be a character, a menu item ID, or a notification constant.

-> nameP  Pointer to the text to display in the status message if the user taps the button. If NULL, the text is taken from the menu item that matches the ID or shortcut character contained in result, if a match is found.

If you supply a text buffer for this parameter, **MenuCmdBarAddButton** makes a copy of the buffer.

**Result**  Returns 0 upon success, or one of the following error codes:

```
menuErrOutOfMemory  There is not enough memory available to perform the operation.

menuErrTooManyItems  The command toolbar already has the maximum number of buttons allowed (currently 8).
```

**Comments**  Call this function in response to a **menuCmdBarOpenEvent** or to the notification **sysNotifyMenuCmdBarOpenEvent**. Both of these signal that the user has entered the command keystroke and the command toolbar is about to open. Your response should be to add buttons to the toolbar and to return false, indicating that you have not completely handled the event.

The **sysNotifyMenuCmdBarOpenEvent** notification is intended to be used only by shared libraries, system extensions, and other code resources that do not use an event loop. If you’re writing an application, always respond to the event instead of the notification; an application should only add buttons to the toolbar if it is the...
current application. If you register for the notification, you receive it each time the command toolbar is displayed, whether your application is active or not.

Note that the command toolbar is allocated each time it is opened and is deallocated when it is erased from the screen.

There is a limited amount of space in which to display buttons on the command toolbar. You should limit the number of buttons to four or five. The maximum allowed by the system is eight, but you should leave space for the status message that appears after the user chooses an action. Buttons should be contextual; for example, the field code only displays a paste button if there is text on the clipboard. Bitmaps for the buttons should be 16 X 13 pixels.

If a field has focus when the command toolbar is opened, the field manager adds buttons for cut, copy, paste, and undo. If your application does not want this default behavior, set the preventFieldButtons field in the menuCmdBarOpenEvent structure to true. (Note that there is no way to prevent the field buttons from being drawn from within a notification handler.)

The following bitmaps for command toolbar buttons are defined in UIResources.h. The system and the built-in applications use these bitmaps to represent the commands listed in the table. Your application should also use them if it performs the same actions. If you use any of these buttons, add them in the order shown from right to left. (For example, BarDeleteBitmap, if used, should always be the rightmost button.)

<table>
<thead>
<tr>
<th>Bitmap</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>BarDeleteBitmap</td>
<td>Delete record.</td>
</tr>
<tr>
<td>BarPasteBitmap</td>
<td>Paste clipboard contents at insertion point.</td>
</tr>
<tr>
<td>BarCopyBitmap</td>
<td>Copy selection.</td>
</tr>
<tr>
<td>BarCutBitmap</td>
<td>Cut selection.</td>
</tr>
<tr>
<td>BarUndoBitmap</td>
<td>Undo previous action.</td>
</tr>
<tr>
<td>BarSecureBitmap</td>
<td>Show Security dialog.</td>
</tr>
</tbody>
</table>
It is best to add buttons on the left side. If you add buttons to the right, this function moves all existing buttons over one position to the left. You can also specify an exact position for the \texttt{where} parameter. The positions are numbered from right to left with the rightmost position being 1. If you specify an exact position, the function leaves space for the other buttons. For example, if you specify position 3 and there are no buttons displayed at positions 1 and 2, there will be blank spots to the right of your button.

The \texttt{result} and \texttt{resultType} parameters specify what the result should be if the user taps the button. \texttt{result} contains the actual data, and \texttt{resultType} contains a constant that specifies the type of data in \texttt{result}. Typically, the result is to enqueue a \texttt{menuEvent}. In this case, \texttt{resultType} is \texttt{menuCmdBarResultMenuIItem} and the \texttt{result} is the ID of the menu item that should be included in the event.

You may also specify the shortcut character instead of the menu ID; however, doing so is inefficient. When \texttt{result} is a shortcut character, the \texttt{MenuHandleEvent} function enqueues a \texttt{keyDownEvent} with the character in \texttt{result}. During the next cycle of the event loop, \texttt{MenuHandleEvent} enqueues a \texttt{menuEvent} in response to the \texttt{keyDownEvent}. Thus, it is better to have your button enqueue the \texttt{menuEvent} directly.

If you call \texttt{MenuCmdBarAddButton} outside of an application, you might not know of any menu items in the active menu (unless your code has added one using \texttt{MenuAddItem}). In this case, specify a notification to be broadcast. The notification is broadcast at the top of the next event loop, and it must contain no custom data. (Applications may also use the notification result type.)

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{Bitmap} & \textbf{Command} \\
\hline
BarBeamBitmap & Beam current record. \\
BarInfoBitmap & Show Info dialog (Launcher). \\
\hline
\end{tabular}
\end{table}

**Compatibility**

Implemented only if \texttt{3.5 New Feature Set} is present.

**See Also**

\texttt{MenuCmdBarDisplay}, \texttt{MenuCmdBarGetButtonData}
**MenuCmdBarDisplay**

**Purpose**
Displays the command toolbar.

**Declared In**
Menu.h

**Prototype**
```c
void MenuCmdBarDisplay (void)
```

**Parameters**
None

**Result**
Returns nothing.

**Comments**
This function displays the command toolbar when the user enters the command keystroke. You normally do not call this function in your own code. The form manager calls it at the end of its handling of `menuCmdBarOpenEvent`.

**Compatibility**
Implemented only if [3.5 New Feature Set](#) is present.

**See Also**
`MenuCmdBarAddButton`, `MenuCmdBarGetButtonData`

---

**MenuCmdBarGetButtonData**

**Purpose**
Gets the data for a given command button.

**Declared In**
Menu.h

**Prototype**
```c
Boolean MenuCmdBarGetButtonData
    (Int16 buttonIndex, UInt16 *bitmapIdP,
     MenuCmdBarResultType *resultTypeP,
     UInt32 *resultP, Char *nameP)
```

**Parameters**
- `->buttonIndex` Index of the button for which you want to obtain information. Buttons are ordered from right to left, with the rightmost button at index 0.
Menus
Menu Functions

<- bitmapIdP | The resource ID of the bitmap displayed on the button. Pass NULL if you don’t want to retrieve this value.

<- resultTypeP | The type of action this button takes. Pass NULL if you don’t want to retrieve this value.

<- resultP | The result of tapping the button. Pass NULL if you don’t want to retrieve this information.

<- nameP | The text displayed in the status message when this button is tapped. Pass NULL if you don’t want to retrieve this information. If not NULL, nameP must point to a string of menuCmdBarMaxTextLength size.

Result | Returns true if the information was retrieved successfully, false if there is no command toolbar or if there is no button at buttonIndex.

Comments | You can use this function to retrieve information about the buttons that are displayed on the command toolbar. If the command toolbar has not yet been initialized, this function returns false.

Note that the command toolbar is allocated when the user enters the command keystroke and deallocated when MenuEraseStatus is called. Thus, the only logical place to call MenuCmdBarGetButtonData is in response to a menuCmdBarOpenEvent or sysNotifyMenuCmdBarOpenEvent notification.

Compatibility | Implemented only if 3.5 New Feature Set is present.

See Also | MenuCmdBarDisplay, MenuCmdBarAddButton
Menus
Menu Functions

MenuDispose

Purpose
Releases any memory allocated to the menu and the command status and restore any saved bits to the screen.

Declared In Menu.h

Prototype void MenuDispose (MenuBarType *menuP)

Parameters
- menuP Pointer to the menu object to dispose. (See MenuBarType.) If NULL, this function returns immediately.

Result Returns nothing.

Comments Most applications do not need to call this function directly. MenuDispose is called by the system when the form that contains the menu is no longer the active form, when the form that contains the menu is freed, and when FrmSetMenu is called to change the form’s menu bar.

See Also MenuInit, MenuDrawMenu

MenuDrawMenu

Purpose Draws the current menu bar and the last pull-down that was visible.

Declared In Menu.h

Prototype void MenuDrawMenu (MenuBarType *menuP)

Parameters
- menuP Pointer to a MenuBarType. Must not be NULL.

Result Returns nothing.

Comments Most applications do not need to call this function directly. MenuHandleEvent calls MenuDrawMenu when the user taps the
Menu silk-screen button (or taps the form’s title on Palm OS 3.5 and higher).

The menu bar and the pull-down menu are drawn in front of all the application windows. The state of the insertion point, the bits that are obscured by the menu bar and the pull-down menu, and the currently active window are saved before the menu is drawn. These are all restored when the menu is erased.

A menu keeps track of the last pull-down menu displayed for as long as the menu is active. A menu loses its active status under these conditions:

- When `FrmSetMenu` is called to change the active menu on the form.
- When a new form, even a modal form or alert panel, becomes active.

Suppose a user selects your application’s About item from the Options menu then clicks the OK button to return to the main form. When the About dialog is displayed, it becomes the active form, which causes the main form’s menu state to be erased. This menu state is not restored when the main form becomes active again. The next time `MenuDrawMenu` is called (that is, the next time the user taps the Menu silk-screen button), the menu bar is displayed as it was before, and the first pull-down menu listed in the menu bar is displayed instead of the Options pull-down menu.

See Also  
`MenuInit`, `MenuDispose`
MenuEraseStatus

Purpose
Erases the menu command status.

Declared In
Menu.h

Prototype
void MenuEraseStatus (MenuBarType *menuP)

Parameters
- menuP   Pointer to a MenuBarType, or NULL for the current menu.

Result
Returns nothing.

Comments
When the user selects a menu command using the command keystroke, the command toolbar or status message is displayed at the bottom of the screen. MenuEraseStatus erases the toolbar or status message.

Under most circumstances, you do not need to call this function explicitly—just let the current menu command status remove itself automatically. Otherwise, you may cause text to be erased before the user has a chance to see it.

You need to call MenuEraseStatus explicitly only if the command toolbar covers something that is going to be changed by the menu command the user has selected. For example, if the user selects a command that displays a new form, call MenuEraseStatus before executing the command. Also, if the command performs some drawing in the lower portion of the window, call MenuEraseStatus before performing the drawing function.

Compatibility
The exact behavior when a menu shortcut character is entered depends on which version of the operating system is running. In versions prior to release 3.5, the system displays the string “Command:” in the lower-left portion of the screen when the user enters the Graffiti command keystroke.

In Palm OS 3.5 and higher, entering the Graffiti command keystroke displays the command toolbar. This toolbar is the entire width of the screen and it displays buttons that the user can tap instead of entering another keystroke. If the user taps a button or enters a
character that matches a shortcut character for an item on the active menu, a status message is displayed in the toolbar while the command is executed. Calling MenuEraseStatus on Palm OS 3.5 and higher deallocates the command toolbar data structure as well as erasing the command toolbar from the screen.

Because the command toolbar takes up more of the display than the pre-Palm OS 3.5 status message, you may find you need to call MenuEraseStatus more frequently in Palm OS 3.5 than in earlier versions.

See Also  MenuInit

MenuGetActiveMenu

Purpose  Returns a pointer to the currently active menu.

Declared In  Menu.h

Prototype  MenuBarType *MenuGetActiveMenu (void)

Parameters  None.

Result  Returns a pointer to the currently active menu, or NULL if there is none.

Comments  An active menu is not necessarily visible on the screen. A menu might be active but not visible, for example, if a command shortcut has been entered. In general, a form’s menu becomes active the first time a keyDownEvent with a vchrMenu or vchrCommand is generated, and it remains active until a new form (including a modal form or alert panel) is displayed or until FrmSetMenu is called to change the form’s menu.

Applications that perform custom drawing to a window often check to see if the menu is visible to ensure that they don’t draw on top of the menu. See “Checking Menu Visibility” on page 107 of the Palm
See Also MenuHandleEvent, MenuSetActiveMenu

MenuHandleEvent

Purpose Handles events in the current menu. This routine handles two types of events, penDownEvent and keyDownEvent.

Declared In Menu.h

Prototype Boolean MenuHandleEvent (MenuBarType *menuP, EventType *event, UInt16 *error)

Parameters
- menuP Pointer to a MenuBarType data structure.
- event Pointer to an EventType structure.
- error Error (or 0 if no error). Currently this function always sets error to zero.

Result Returns true if the event is handled; that is, if the event is a penDownEvent within the menu bar or the menu, or the event is a keyDownEvent that the menu supports. Returns false on any other event.

Comments When a penDownEvent is received in the menu bar, MenuHandleEvent tracks the pen until it comes up. If the pen comes up within the bounds of the menu bar, the selected title is inverted and the appropriate pull-down menu is drawn. Any previous pull-down menu is erased. If the pen comes up outside of the menu bar and pull-down menu, the menu is erased.

When a penDownEvent is received in a pull-down menu, MenuHandleEvent tracks the pen until it comes up. If the pen comes up within the bounds of the menu, a menuEvent containing the resource ID of the selected menu item is added to the event queue. If the pen comes up outside of the bounds of the menu and menu bar, the menu is erased.
If a `keyDownEvent` is received with a `vchrMenu`, the menu is drawn if it is not visible and erased if it is visible.

If a `keyDownEvent` is received with a `vchrCommand`, a command shortcut is in progress. Command shortcuts are handled differently depending on which version of Palm OS is running. On versions earlier than 3.5, the next `keyDownEvent` is checked to see if it is a valid menu shortcut. If so, a `menuEvent` is added to the event queue.

If a `keyDownEvent` is received with a `vchrCommand` on Palm OS 3.5 and higher, `MenuHandleEvent` enqueues a `menuCmdBarOpenEvent` if the command toolbar is not already open. The `menuCmdBarOpenEvent` provides a chance for applications to add their own buttons to the command toolbar. The next event might be either a `keyDownEvent` with a character that completes the shortcut or a `penDownEvent` on one of the buttons on the toolbar. The `keyDownEvent` is processed as with the earlier releases— if it is a valid menu shortcut, a `menuEvent` is added to the event queue. If the next event is a `penDownEvent`, the pen is tracked until it comes up. If the pen comes up within the bounds of a button, the appropriate action is taken. See the description of `MenuCmdBarAddButton` for more information.

In Palm OS version 3.5 or higher, if either the `vchrMenu` or the `vchrCommand` event causes a menu to be activated and initialized for the first time, a `menuOpenEvent` containing the reason the menu was initialized (`menuButtonCause` for a `vchrMenu` or `menuCommandCause` for a `vchrCommand`) is added to the event queue, and then the current event is added after it.
**MenuHideItem**

**Purpose**
Makes the specified menu item hidden.

**Declared In**
Menu.h

**Prototype**
Boolean MenuHideItem (UInt16 id)

**Parameters**
- `id` ID of the menu item to hide.

**Result**
Returns `true` if the hidden attribute of the specified item was successfully enabled, `false` otherwise.

**Comments**
You should call this function only in response to a `menuOpenEvent`. This event is generated when the menu is first made active. In general, a form’s menu becomes active the first time a `keydownEvent` with a `vchrMenu` or `vchrCommand` is generated, and it remains active until a new form (including a modal form or alert panel) is displayed or until `FrmSetMenu` is called to change the form’s menu. Palm OS user interface guidelines discourage adding or hiding menu items at any time other than when the menu is first made active.

**Compatibility**
Implemented only if 3.5 New Feature Set is present.

**See Also**
MenuShowItem
MenuInit

**Purpose**
Loads a menu resource from a resource file.

**Declared In**
Menu.h

**Prototype**
MenuBarType *MenuInit (UInt16 resourceId)

**Parameters**
-> resourceId  ID that identifies the menu resource.

**Result**
Returns the pointer to a memory block allocated to hold the menu resource (a pointer to a MenuBarType).

**Comments**
The menu is not usable until MenuSetActiveMenu is called.

Typically, you do not need to call this function directly. A form stores the resource ID of the menu associated with it and initializes that menu as necessary. If you want to change the form’s menu, call FrmSetMenu, which handles disposing of the form’s current menu, associating the new menu with the form, and initializing when needed.

**See Also**
MenuSetActiveMenu, MenuDispose

MenuSetActiveMenu

**Purpose**
Sets the current menu.

**Declared In**
Menu.h

**Prototype**
MenuBarType *MenuSetActiveMenu (MenuBarType *menuP)

**Parameters**
-> menuP  Pointer to the memory block that contains the new menu, or NULL for none.

**Result**
Returns a pointer to the menu that was active before the new menu was set, or NULL if no menu was active.
Comments  This function sets the active menu but does not associate it with a form. It’s recommended that you call FrmSetMenu instead of MenuSetActiveMenu. FrmSetMenu sets the active menu, frees the old menu, and associates the newly active menu with the form, which means the menu will be freed when the form is freed.

See Also  MenuGetActiveMenu

**MenuSetActiveMenuRscID**

**Purpose**  Sets the current menu by resource ID.

**Declared In**  Menu.h

**Prototype**  void MenuSetActiveMenuRscID (UInt16 resourceId)

**Parameters**  -> resourceId  Resource ID of the menu to be made active.

**Result**  Returns nothing.

Comments  This function is similar to MenuSetActiveMenu except that you pass the menu’s resource ID instead of a pointer to a menu structure. It is used as an optimization; with MenuSetActiveMenu, you must initialize the menu before making it active. Potentially, the application may exit before the menu is needed, making this memory allocation unnecessary. MenuSetActiveMenuRscID simply stores the resource ID. The next time a menu is requested, the menu is initialized from this resource.

It’s recommended that you call FrmSetMenu instead of calling this function for the reasons given in MenuSetActiveMenu.

**Compatibility**  Implemented only if “2.0 New Feature Set” is present.
**MenuShowItem**

**Purpose**  Makes the specified menu item visible.

**Declared In**  Menu.h

**Prototype**  Boolean MenuShowItem (UInt16 id)

**Parameters**  
- **id**  ID of the menu item to display.

**Result**  Returns true if the hidden attribute of the specified item was successfully disabled, false otherwise.
Comments You should call this function only in response to a menuOpenEvent. This event is generated when the menu is first made active. In general, a form’s menu becomes active the first time a keyDownEvent with a vchrMenu or vchrCommand is generated, and it remains active until a new form (including a modal form or alert panel) is displayed or until FrmSetMenu is called to change the form’s menu. Palm OS user interface guidelines discourage adding or hiding menu items at any time other than when the menu is first made active.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also MenuHideItem
Private Records

This chapter describes the private records API as declared in PrivateRecords.h. It discusses the following topics:

- Private Record Data Structures
- Private Record Functions

 Private Record Data Structures

**privateRecordViewEnum**

The privateRecordViewEnum enumerated type provides the available choices for displaying private records.

```c
typedef enum privateRecordViewEnum {
    showPrivateRecords = 0x00,
    maskPrivateRecords,
    hidePrivateRecords
} privateRecordViewEnum;
```

**Value Descriptions**

- **showPrivateRecords**  
  Display private records in the user interface.

- **maskPrivateRecords**  
  Show a shaded rectangle in place of a private record.

- **hidePrivateRecords**  
  Hide private records and provide no indication in the user interface that they exist.
Private Record Functions

SecSelectViewStatus

Purpose
Display a form that allows the user to select whether to hide, show, or mask private records.

Declared In
PrivateRecords.h

Prototype
privateRecordViewEnum SecSelectViewStatus (void)

Parameters
None

Result
Returns a constant that indicates which option the user selected. See privateRecordViewEnum.

Comments
This function displays a dialog that allows users to change the preference prefShowPrivateRecords, which controls how private records are displayed.

When the user taps the OK button in this dialog, SecVerifyPW is called to see if the user changed the preference setting and, if so, to prompt the user to enter the appropriate password.

After calling this function, your code should check the return value or the value of prefShowPrivateRecords and mask, display, or hide the private records accordingly. See the description of TblSetRowMasked for a partial example.

Compatibility
Implemented only if 3.5 New Feature Set is present.
SecVerifyPW

Purpose
Display a password dialog, verify the password, and change the private records preference.

Declared In
PrivateRecords.h

Prototype
Boolean SecVerifyPW
(privateRecordViewEnum newSecLevel)

Parameters
-> newSecLevel  The security level (display, hide, or mask) selected on the private records dialog.

Result
Returns true if the prefShowPrivateRecords preference was successfully changed, false if not.

Comments
This function checks newSecLevel against the current value for the preference. If the two values differ and newSecLevel indicates a decrease in security, a dialog is displayed prompting the user to enter a password. (Hidden is considered the most secure, followed by masked. Showing private records is considered the least secure.) If the password is entered successfully, the preference is changed.

This function also displays an alert message if the security level has changed to either hidden or masked.

Compatibility
Implemented only if 3.5 New Feature Set is present.
Progress Manager

This chapter provides reference material for the Progress Manager.

- Progress Manager Functions
- Application-Defined Functions

The header file Progress.h declares the API that this chapter describes. For more information on the progress manager, see the section “Progress Dialogs” in the Palm OS Programmer’s Companion, vol. I.

Progress Manager Functions

PrgHandleEvent

**Purpose**
Handles events related to the active progress dialog.

**Declared In**
Progress.h

**Prototype**
Boolean PrgHandleEvent (ProgressPtr prgGP, EventType *eventP)

**Parameters**
- **-> prgGP**
  Pointer to a progress structure created by PrgStartDialog.
- **-> eventP**
  Pointer to an event. You can pass a NULL event to cause this function to immediately call your PrgCallbackFunc function and then update the dialog (for example, after you call PrgUpdateDialog).

**Result**
Returns true if the system handled the event. If it returns false, you should check if the user canceled the dialog by calling PrgUserCancel.
Comments

Use this function instead of SysHandleEvent when you have a progress dialog. PrgHandleEvent internally calls SysHandleEvent as needed.

Note that the auto power-off feature of the system is automatically disabled when you use this function, unless the dialog is just displaying an error. This function also prevents appStopEvent events.

If an update to the dialog is pending (from a call to PrgUpdateDialog, for example) this function handles that event and causes the dialog to be updated. As part of this process, the textCallback function you specified in your call to PrgStartDialog is called. Your textCallback function should set the textP buffer in the PrgCallbackData structure with the new message to be displayed in the progress dialog. Optionally, you can also set the bitmapId field to include an icon in the update dialog. For more information about the textCallback function, see the section “Application-Defined Functions.”

Compatibility

Implemented only if 3.0 New Feature Set is present.

See Also

PrgStartDialog, PrgStopDialog, PrgUpdateDialog, PrgUserCancel

PrgStartDialog

Purpose

Displays a progress dialog that can be updated.

Declared In

Progress.h

Prototype

ProgressPtr PrgStartDialog (const Char *title, PrgCallbackFunc textCallback, void *userDataP)

Parameters

-> title Pointer to a title for the progress dialog. Must be a null-terminated string that is no longer than progressMaxTitle (20).

-> textCallback Pointer to a callback function that supplies the text and icons for the current progress state. See PrgCallbackFunc.
-> userDataP A pointer to data that you need to access in the callback function. This address gets passed directly to your function.

**Result**
A pointer to a progress structure. This pointer must be passed to other progress manager functions and **must** be released by calling `PrgStopDialog`. NULL is returned if the system is unable to allocate the progress structure.

**Comments**
The dialog created by this function can be updated by another process via the `PrgUpdateDialog` function. The dialog can contain a Cancel or OK button. The initial dialog defaults to stage 0 and calls the `textCallback` function to get the initial text and icon data for the progress dialog.

This function saves the screen bits behind the progress dialog, and these are restored when you call `PrgStopDialog`. Because of this, you should minimize changes to the screen while the progress dialog is displayed, otherwise, the restored bits may not match with what is currently being displayed.

**Compatibility**
This version of the function is available only if 3.2 New Feature Set is present. On earlier systems, `PrgStartDialog` has the prototype shown for `PrgStartDialogV31`.

**See Also**
`PrgHandleEvent`, `PrgStopDialog`, `PrgUpdateDialog`, `PrgUserCancel`
PrgStartDialogV31

**Purpose**
Displays a progress dialog that can be updated.

**Declared In**
Progress.h

**Prototype**
ProgressPtr PrgStartDialogV31 (const Char *title, PrgCallbackFunc textCallback)

**Parameters**
- `title` Pointer to a title for the progress dialog. Must be a null-terminated string that is no longer than progressMaxTitle (20).
- `textCallback` Pointer to a callback function that supplies the text and icons for the current progress state. See PrgCallbackFunc.

**Result**
A pointer to a progress structure. This pointer must be passed to other progress manager functions and must be released by calling PrgStopDialog. NULL is returned if the system is unable to allocate the progress structure.

**Compatibility**
This function corresponds to version of PrgStartDialog available on Palm OS® 3.0 and Palm OS 3.1.

**See Also**
PrgHandleEvent, PrgStopDialog, PrgUpdateDialog, PrgUserCancel
**PrgStopDialog**

**Purpose** Releasing memory used by the progress dialog and restores the screen to its initial state.

**Declared In** Progress.h

**Prototype**

```c
void PrgStopDialog (ProgressPtr prgP, Boolean force)
```

**Parameters**

- `prgP` Pointer to a progress structure created by `PrgStartDialog`.
- `force` true removes the progress dialog immediately, false causes the system to wait until the user confirms an error, if one is displayed.

**Result** Returns nothing.

**Comments** If the progress dialog is in a state where it is displaying an error message to the user, this function normally waits for the user to confirm the dialog before it removes the dialog. If you specify true for the `force` parameter, this causes the system to remove the dialog immediately.

**Compatibility** Implemented only if 3.0 New Feature Set is present.

**See Also** `PrgHandleEvent`, `PrgStartDialog`, `PrgUpdateDialog`, `PrgUserCancel`
**PrgUpdateDialog**

**Purpose** Updates the status of the current progress dialog.

**Declared In** Progress.h

**Prototype**
```c
void PrgUpdateDialog (ProgressPtr prgGP,
UInt16 err, UInt16 stage, const Char *messageP,
Boolean updateNow)
```

**Parameters**
- `-> prgGP` Pointer to a progress structure created by PrgStartDialog.
- `-> err` If non-zero, causes the dialog to go into error mode, to display an error message with only an OK button.
- `-> stage` Current stage of progress. The callback function can use this to determine the correct string to display in the updated dialog.
- `-> messageP` Extra text that may be useful in displaying the progress for this stage. Used by the callback function, which can append it to the base message that is based on the stage.
- `-> updateNow` If true, the dialog is immediately updated. Otherwise, the dialog is updated on the next call to PrgHandleEvent.

**Result** Returns nothing.

**Comments** For more information about how the parameters are used and the callback function, see the section “Application-Defined Functions.”

**Compatibility** Implemented only if 3.0 New Feature Set is present.

**See Also** PrgHandleEvent, PrgStartDialog, PrgStopDialog, PrgUserCancel
**PrgUserCancel**

**Purpose**
Macro that returns `true` if the user cancelled the process via the progress dialog.

**Declared In**
Progress.h

**Prototype**
`PrgUserCancel (prgP)`

**Parameters**
- `prgP` Pointer to a progress structure (ProgressPtr) created by `PrgStartDialog`.

**Result**
Returns the value of the `cancel` field in the progress structure (as a UInt16).

**Comments**
This is a macro you can use to check if the user cancelled the process. If the user did cancel, you can change the progress dialog text to something like “Cancelling,” or “Disconnecting,” or whatever is appropriate for your application. Then you should cancel the process, end the communication session, or do whatever processing is necessary.

**Compatibility**
Implemented only if 3.0 New Feature Set is present.

**See Also**
PrgHandleEvent, PrgStartDialog, PrgStopDialog, PrgUpdateDialog
Application-Defined Functions

PrgCallbackFunc

**Purpose**
Supplies the text and icons for the current progress state.

**Declared In**
Progress.h

**Prototype**

```
Boolean (*PrgCallbackFunc)(PrgCallbackDataPtr cbP)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cbP</td>
<td>A pointer to a PrgCallbackData structure. See below.</td>
</tr>
</tbody>
</table>

**Result**

Returns true if the progress dialog should be updated using the values you specified in the PrgCallbackData structure. If you specify false, the dialog is still updated, but with default status messages. (Returning false is not recommended.)

**Comments**

This is a callback function that you specify when you call PrgStartDialog. The callback function is called by PrgHandleEvent when it needs current progress information to display in the progress dialog.

The system passes this function one parameter, a pointer to a PrgCallbackData structure. Here are the important fields in that data structure (note that -> indicates you set the field in the textCallback function):

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stage</td>
<td>Current stage (passed from PrgUpdateDialog).</td>
</tr>
<tr>
<td>textP</td>
<td>Buffer to hold the text to display in the updated dialog. You might want to look up a message in a resource file, based on the value in the stage field. Also, you should append the additional text in the message field, to form the full string to display. Be sure to include a null terminator at the end of the string you return, and don’t exceed the length in textLen.</td>
</tr>
</tbody>
</table>
<- UInt16 textLen
Maximum length of the text buffer textP. Note that this value is set for you by the caller. Be careful not to exceed this length in textP.

<- Char *message
Additional text to display in the dialog (from the messageP parameter to PrgUpdateDialog). This should be no longer than progressMaxMessage (128).

<- Err error
Current error (passed from the err parameter to PrgUpdateDialog).

-> UInt16 bitmapId
Resource ID of the bitmap to display in the progress dialog, if any.

<- UInt16 canceled:1
true if user has pressed the cancel button.

<- UInt16 showDetails:1
true if user pressed the down arrow button on the Palm™ device for more details. (Because this is a non-standard user interface technique, you shouldn’t use this feature to display details that users need under normal conditions. It’s more for debugging purposes.)

-> UInt16 textChanged:1
If true, then update text (defaults to true). You can set this to false to avoid an update to the text.

<- UInt16 timedOut:1
true if update caused by a timeout.
<-> UInt32 timeout
    Timeout in ticks to force next update. After this number of ticks, an update is automatically triggered (which sets the timedOut flag). You can use this feature to do a simple animation effect. Note that you must set the timeout for EvtGetEvent to a value that is equal to or less than this value, otherwise you won’t get update events as frequently as you expect.

-> UInt16 delay:1
    If true, delay for one second after updating the dialog. Use this value when you are displaying the final progress message so that users have a chance to see the message before the dialog closes. This field is available only if 3.2 New Feature Set is present.

<- void *userDataP
    A pointer to any application-defined data that the function needs to access. You specify this pointer as a parameter to PrgStartDialog if the callback function needs to access some application data but does not have access to application globals. This field is available only if 3.2 New Feature Set is present.

In this function, you should set the value of the textP buffer to the string you want to display in the progress dialog when it is updated. You can use the value in the stage field to look up a message in a string resource. You also might want to append the text in the message field to your base string. Typically, the message field would contain more dynamic information that depends on a user selection, such as a phone number, device name, or network identifier, etc.

For example, the PrgUpdateDialog function might have been called with a stage of 1 and a messageP parameter value of a phone number string, “555-1212”. Based on the stage, you might find the string “Dialing” in a string resource, and append the phone number, to form the final text “Dialing 555-1212” that you place in the text buffer textP.
Keeping the static strings corresponding to various stages in a resource makes it easier to localize your application. More dynamic information can be passed in via the `messageP` parameter to `PrgUpdateDialog`.

**NOTE:** This function is called only if the parameters passed to `PrgUpdateDialog` have changed from the last time it was called. If `PrgUpdateDialog` is called twice with exactly the same parameters, the `textCallback` function is called only once.
Progress Manager
Application-Defined Functions
Scroll Bar Data Structures

ScrollBarAttrType

The ScrollBarAttrType bit field defines a scroll bar’s visible characteristics.

typedef struct {
    UInt16 usable : 1;
    UInt16 visible : 1;
    UInt16 hilighted : 1;
    UInt16 shown : 1;
    UInt16 activeRegion: 4;
} ScrollBarAttrType;
Scroll Bars
Scroll Bar Data Structures

Field Descriptions

usable If not set, the scroll bar object is not considered part of the current interface of the application, and it doesn’t appear on screen.

visible If set, the scroll bar is allowed to be displayed on the screen. If both visible and shown are true, then the scroll bar is actually displayed on the screen.

highlighted true if either the up arrow or the down arrow is highlighted.

shown Set if the scroll bar is visible and if maxValue > minValue. (See ScrollBarType.)

activeRegion The region of the scroll bar that is receiving the pen down events. Possible values are:

sclUpArrow The up arrow.
sclDownArrow The down arrow.
sclUpPage The region between the scroll car and the up arrow.
sclDownPage The region between the scroll car and the down arrow.
sclCar The scroll car.

ScrollBarPtr

The ScrollBarPtr type defines a pointer to a ScrollBarType structure.

typedef ScrollBarType *ScrollBarPtr;

You pass the ScrollBarPtr as an argument to all scroll bar functions. You can obtain the ScrollBarPtr using the function FrmGetObjectPtr in this way:
scrollBarPtr = FrmGetObjectPtr(frm, 
        FrmGetObjectIndex(frm, scrollBarID));

where scrollBarID is the resource ID assigned when you created 
the scroll bar.

**ScrollBarType**
The ScrollBarType represents a scroll bar.

**WARNING!** PalmSource, Inc. does not support or provide 
backward compatibility for the ScrollBarType structure. Never 
access its structure members directly, or your code may break in 
future versions. Use the information below for debugging 
purposes only.

```c
typedef struct {
    RectangleType         bounds;
    UInt16                id;
    ScrollBarAttrType     attr;
    Int16                 value;
    Int16                 minValue;
    Int16                 maxValue;
    Int16                 pageSize;
    Int16                 penPosInCar;
    Int16                 savePos;
} ScrollBarType;
```

Your code should treat the ScrollBarType structure as opaque. 
Use the functions described in this chapter to retrieve and set each 
value. Do not attempt to change structure member values directly.
Field Descriptions

bounds
Position (using absolute coordinates) and size (in pixels) of the scroll bar on the screen. For example, to access the bounds of an object in a form whose ID is kObjectID:
{
    RectangleType rect;
    FormPtr formP = FrmGetActiveForm();
    FrmGetObjectBounds(formP, FrmGetObjectIndex(formP, kObjectID), &rect);
}

id
ID value you specified when you created the scroll bar object.

attr
Scroll bar’s attributes. See ScrollBarAttrType.

value
Current value of the scroll bar. This value is used to determine where to position the scroll car (the dark region in the scroll bar that indicates the position in the document). Access with SclGetScrollBar.

The number given is typically a number relative to minValue and maxValue. These values have nothing to do with any physical characteristics of the object that the scroll bar is attached to, such as the number of lines in the object.

This value is typically set to 0 initially and then adjusted programmatically with SclSetScrollBar.

minValue
Minimum value. When value equals minValue, the scroll car is positioned at the very top of the scrolling region. This value is typically 0. Access with SclGetScrollBar.
**Scroll Bars**

*Scroll Bar Data Structures*

- **maxValue**
  - Maximum value. When `value` equals `maxValue`, the scroll car is positioned at the very bottom of the scrolling region. This value is typically set to 0 initially and then adjusted programmatically with `SclSetScrollBar`. Access with `SclGetScrollBar`.

- **pageSize**
  - Number of lines to scroll when user scrolls one page. Access with `SclGetScrollBar`.

- **penPosInChar**
  - Used internally.

- **savePos**
  - Used internally.
Scroll Bar Resources

The Scroll Bar Resource (tSCL) represents a scroll bar.

Scroll Bar Functions

**SclDrawScrollBar**

**Purpose**
Draw a scroll bar.

**Declared In**
ScrollBar.h

**Prototype**
```c
void SclDrawScrollBar (ScrollBarType *bar)
```

**Parameters**
- `-> bar` Pointer to a scroll bar structure (see `ScrollBarType`).

**Result**
Returns nothing.

**Comments**
This function is called internally by `SclSetScrollBar` and `FrmDrawForm`. You rarely need to call it yourself.

**Compatibility**
Implemented only if 2.0 New Feature Set is present.

**SclGetScrollBar**

**Purpose**
Retrieve a scroll bar’s current position, its range, and the size of a page.

**Declared In**
ScrollBar.h

**Prototype**
```c
void SclGetScrollBar (const ScrollBarType *bar, Int16 *valueP, Int16 *minP, Int16 *maxP, Int16 *pageSizeP)
```

**Parameters**
- `-> bar` Pointer to a scroll bar structure (see `ScrollBarType`).
<-valueP  A value representing the scroll car’s current position. (The scroll car is the dark region that indicates the position in the document.)

<-minP  A value representing the top of the user interface object.

<-maxP  A value representing the bottom of the user interface object.

<-pageSizeP  Pointer to size of a page (used when page scrolling).

**Result**

Returns the scroll bar’s current values in valueP, minP, maxP, and pageSizeP.

**Comments**

You might use this function immediately before calling `SclSetScrollBar` to update the scroll bar. `SclGetScrollBar` returns the scroll bar’s current values, which you can then adjust as necessary and pass to `SclSetScrollBar`.

**Compatibility**

Implemented only if [2.0 New Feature Set](#) is present.

**See Also**

`SclSetScrollBar`

### SclHandleEvent

**Purpose**

Handles events that affect a scroll bar.

**Declared In**

ScrollBar.h

**Prototype**

```c
Boolean SclHandleEvent (ScrollBarType *bar, const EventType *event)
```

**Parameters**

- `-> bar`  Pointer to a scroll bar structure (see `ScrollBarType`).
- `-> event`  Pointer to an event (`EventType`).

**Result**

Returns true if the event was handled.
Comment

When a penDownEvent occurs, the scroll bar sends an sclEnterEvent to the event queue.

When an sclEnterEvent occurs, the scroll bar determines what its new value should be based on which region of the scroll bar is receiving the pen down events. It then sends either an sclRepeatEvent or an sclExitEvent to the event queue.

When the user holds and drags the scroll bar with the pen, the scroll bar sends a sclRepeatEvent. Applications that implement dynamic scrolling should catch this event and move the text each time one arrives.

When the user releases the pen from the scroll bar, the scroll bar sends a sclExitEvent. Applications that implement non-dynamic scrolling should catch this event and move the text when sclExitEvent arrives. Applications that implement dynamic scrolling can ignore this event.

Compatibility

Implemented only if 2.0 New Feature Set is present.

SclSetScrollBar

Purpose

Set the scroll bar’s current position, its range, and the size of a page. If the scroll bar is visible and its minimum and maximum values are not equal, it’s redrawn.

Declared In

ScrollBar.h

Prototype

void SclSetScrollBar (ScrollBarType *bar, Int16 value, Int16 min, Int16 max, Int16 pageSize)

Parameters

- bar
  Pointer to a scroll bar structure (see ScrollBarType).
- value
  The position the scroll car should move to. (The scroll car is the dark region that indicates the position in the document.)
- min
  Minimum value.
- max
  Maximum value.
-> pageSize  
Number of lines of text that can be displayed on a screen at one time (used when page scrolling).

Result  
Returns nothing. May display a fatal error message if the min parameter is greater than the max parameter.

Comments  
Call this function when the user adds or deletes text in a field or when a table row is added or deleted.

For scrolling fields, your application should catch the fldChangedEvent and update the scroll bar at that time.

The max parameter is computed as:

\[
\text{number of lines of text} - \text{page size} + \text{overlap}
\]

where number of lines of text is the total number of lines or rows needed to display the entire object, page size is the number of lines or rows that can be displayed on the screen at one time, and overlap is the number of lines or rows from the bottom of one page to be visible at the top of the next page.

For example, if you have 100 lines of text and 10 lines show on a page, the max value would be 90 or 91, depending on the overlap. So if value is greater than or equal to 90 or 91, the scroll car is at the very bottom of the scrolling region.

You can use the FldGetScrollValues function to compute the values you pass for value, min, and max. For example:

```c
FldGetScrollValues (fld, &scrollPos,
&textHeight, &fieldHeight);

if (textHeight > fieldHeight)
  maxValue = textHeight - fieldHeight;
else if (scrollPos)
  maxValue = scrollPos;
else
  maxValue = 0;

SclSetScrollBar (bar, scrollPos, 0, maxValue, fieldHeight - 1);
```
Scroll Bars
Scroll Bar Functions

In this case, `textHeight` is the number of lines of text and `fieldHeight` is the page size. No lines overlap when you scroll one page. Notice that if the page size is greater than the lines of text, then `max` equals `min`, which means that the scroll bar is not displayed.
For scrolling tables, there is no equivalent to 
FldGetScrollValues. Your application must scroll the table itself 
and keep track of the scroll values. See the 
ListViewUpdateScrollers function in the Memo example 
application (MemoMain.c) for an example of setting scroll bar 
values for a table.

**Compatibility**
Implemented only if [2.0 New Feature Set](#) is present.

**See Also**
[SclGetScrollBar](#)
System Dialogs

This chapter provides reference material for system dialogs declared in the header files FatalAlert.h, Launcher.h, GraffitiReference.h, and GraffitiUI.h.

System Dialog Functions

**SysAppLauncherDialog**

**Purpose** Display the launcher popup, get a choice, ask the system to launch the selected application, clean up, and leave. If there are no applications to launch, nothing happens.

**Declared In** Launcher.h

**Prototype** void SysAppLauncherDialog()

**Parameters** None.

**Result** The system may be asked to launch an application.

**Comments** Typically, this routine is called by the system as necessary. Most applications do not need to call this function themselves.

In Palm OS® version 3.0 and higher the launcher is an application, rather than a popup. This function remains available for compatibility purposes only.

**See Also** SysAppLaunch, the “Application Launcher” section in the Palm OS Programmer’s Companion, vol. I
**SysFatalAlert**

**Purpose**
Display a fatal alert until the user taps a button in the alert.

**Declared In**
FatalAlert.h

**Prototype**
UInt16 SysFatalAlert (const Char *msg)

**Parameters**
msg 
Message to display in the dialog.

**Result**
The button tapped; first button is zero.

**SysGraffitiReferenceDialog**

**Purpose**
Pop up the Graffiti® Reference Dialog.

**Declared In**
GraffitiReference.h

**Prototype**
void SysGraffitiReferenceDialog (ReferenceType referenceType)

**Parameters**
referenceType 
Which reference to display. See GraffitiReference.h for more information.

**Result**
Nothing returned.
Tables

This chapter describes the table API as declared in the header file Table.h. It discusses the following topics:

- Table Data Structures
- Table Constants
- Table Resource
- Table Functions
- Application-Defined Functions

For more information on tables, see the section “Tables” in the Palm OS Programmer’s Companion, vol. I.

Table Data Structures

TableAttrType

The TableAttrType bit field defines the visible characteristics of the table.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the TableAttrType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

typedef struct {
    UInt16 visible:1;
    UInt16 editable:1;
} TableAttrType;
Uint16 editing:1;
Uint16 selected:1;
Uint16 hasScrollBar:1;
Uint16 reserved:11;
} TableAttrType;

Your code should treat the TableAttrType bit field as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change member values directly.

Field Descriptions

visible If set, the table is drawn on the screen. The value of this field is set by TblDrawTable and cleared by TblEraseTable.

editable If set, the user can modify the table. You specify this when you create the table resource.

editing If set, the table is in edit mode. The table is in edit mode while the user edits a text item. The value of this field is returned by TblEditing.

selected If set, the current item (as identified by the TableType fields currentRow and currentColumn) is selected. Use TblGetSelection to retrieve this value.

hasScrollBar If set, the table has a scroll bar. Note that this attribute can only be set programmatically. See TblHasScrollBar.

TableColumnAttrType

The TableColumnAttrType structure defines a column in a table.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the TableColumnAttrType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.
typedef struct {
    Coord     width;
    UInt16    reserved1 : 5;
    UInt16    masked : 1;
    UInt16    editIndicator : 1;
    UInt16    usable : 1;
    UInt16    reserved2 : 8;
    Coord     spacing;
    TableDrawItemFuncPtr drawCallback;
    TableLoadDataFuncPtr loadDataCallback;
    TableSaveDataFuncPtr saveDataCallback;
} TableColumnAttrType;

Your code should treat the TableColumnAttrType structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

**Field Descriptions**

- **width** - The column’s width in pixels. See `TblGetColumnWidth` and `TblSetColumnWidth`.
- **reserved1** - Reserved for future use.
- **masked** - If true and the item’s row also has a masked attribute of true, the table cell is drawn on the screen but is shaded to obscure the information that it contains. See `TblSetColumnMasked`.
- **editIndicator** - If true, items in the column should be highlighted if selected while in edit mode. If false, items in the column should not be highlighted. By default, text field items are highlighted in edit mode, but all other types of items are not highlighted. The default can be overridden with `TblSetColumnEditIndicator`.
**Tables**

*Table Data Structures*

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>usable</td>
<td>If false, the column is not considered part of the current interface of the application, and it doesn't appear on screen. See TblSetColumnUsable.</td>
</tr>
<tr>
<td>reserved2</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>spacing</td>
<td>The spacing in pixels between this column and the next column. See TblGetColumnSpacing and TblSetColumnSpacing.</td>
</tr>
<tr>
<td>drawCallback</td>
<td>Pointer to a function that draws custom items in the column. This function is called during TblDrawTable and TblRedrawTable. See TblSetCustomDrawProcedure.</td>
</tr>
<tr>
<td>loadDataCallback</td>
<td>Pointer to a function that loads data into the column. This function is called during TblDrawTable and TblRedrawTable. See TblSetLoadDataProcedure.</td>
</tr>
<tr>
<td>saveDataCallback</td>
<td>Pointer to a function that saves the data in the column. Called when the focus moves from one table cell to another and when the table loses focus entirely. See TblSetSaveDataProcedure.</td>
</tr>
</tbody>
</table>

**Compatibility**

The masked field is defined only if 3.5 New Feature Set is present.

**TableItemPtr**

A TableItemPtr points to a TableItemType.

```c
typedef TableItemType *TableItemPtr;
```

**TableItemType**

The TableItemType structure defines an item, or cell, within the table.
WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `TableItemType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```c
typedef struct {
    TableItemStyleType   itemType;
    FontID               fontID;
    Int16                intValue;
    Char                 *ptr;
} TableItemType;
```

Your code should treat the `TableItemType` structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

NOTE: None of the table items create memory that you need to free. The table manager handles all of the allocating and deallocating of memory for table items. The only memory you are responsible for freeing is the memory handle containing the text that you want displayed in editable text fields. (See `TableLoadDataFuncType`.)

**Field Descriptions**

- **itemType**  The type of the item, such as a control, a text label, and so on. `TblSetItemStyle` sets this value. The rest of the fields in this struct are either used or not used depending on the `itemType`. See Table 20.1.

- **fontID**  ID of the font used to display the item’s text. `TblGetItemFont` and `TblSetItemFont` retrieve and set this value.
intValue  Integer value of the item. `TblGetItemInt` and `TblSetItemInt` retrieve and set this value.

ptr  Pointer to the item’s text. `TblGetItemPtr` and `TblSetItemPtr` retrieve and set this value. All text items have a maximum of `tableMaxTextItemSize`.

The following table lists the possible values for the `itemType` field, describes how each type is drawn, describes which other fields are used for each `itemType`, and provides special instructions for setting those fields. Note in particular that the `fontID` field is often not used. Instead, certain items are displayed in a standard font. These are noted in the last column of this table.

**Table 20.1 TableItemType fields**

<table>
<thead>
<tr>
<th>ItemType</th>
<th>Description</th>
<th>TableItemType Fields Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkboxTableItem</td>
<td>A checkbox control.</td>
<td>intValue</td>
</tr>
<tr>
<td>customTableItem</td>
<td>Application-defined cell. The height of the item is fixed at 11 pixels.</td>
<td>None. Custom items are drawn using</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the custom drawing function that</td>
</tr>
<tr>
<td></td>
<td></td>
<td>you implement. See <code>TableDrawItemFuncType</code>. If you want, you can store data in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the <code>intValue</code> and <code>ptr</code> fields.</td>
</tr>
<tr>
<td>dateTableItem</td>
<td>Non-editable date in the form <code>month/day</code>, or a dash if the date value is -1. The date is followed by an exclamation point if it has past.</td>
<td>intValue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The <code>intValue</code> field should be a value that can be cast to <code>DateType</code>. <code>DateType</code> is currently defined as a 16-bit number: <code>yyyyyyyymmmmddddd</code> The first 7 bits are the year given as the offset since 1904, the next 4 bits are the month, and the next 5 bits are the day. Dates are always drawn in the current font.</td>
</tr>
</tbody>
</table>
### Table 20.1 TableItemType fields (continued)

<table>
<thead>
<tr>
<th>ItemType</th>
<th>Description</th>
<th>TableItemType Fields Used</th>
</tr>
</thead>
</table>
| labelTableItem          | Non-editable text.        | ptr
|                         |                           | Labels are displayed in the system’s default font along with a terminating colon character (‘:’). Use a customTableItem or tallCustomTableItem if you don't want a colon. |
| numericTableItem        | Non-editable number.      | intValue
|                         |                           | Numbers are displayed in the system’s default bold font.                                 |
| popupTriggerTableItem   | A list with a pop-up trigger. | intValue
|                         |                           | ptr
|                         |                           | intValue is the index of the list item that should be displayed.                         |
|                         |                           | ptr is a pointer to the list.                                                            |
|                         |                           | Lists are displayed in the system’s default font.                                        |
| tallCustomTableItem     | Application-defined cell. | None.                                                                                   |
|                         |                           | Custom items are drawn using the custom drawing function that you implement. See TableDrawItemFuncType. If you want, you can store data in the intValue and ptr fields. |
|                         |                           |                                                                                          |
|                         |                           |                                                                                          |
|                         |                           |                                                                                          |

Palm OS Programmer’s API Reference  425
### Tables

*Table Data Structures*

---

**Table 20.1 TableItem type fields (continued)**

<table>
<thead>
<tr>
<th>ItemType</th>
<th>Description</th>
<th>TableItem type Fields Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>textTableItem</td>
<td>Editable text field.</td>
<td>fontID ptr&lt;br&gt;For this item type, implement the callback function <strong>TableLoadDataFuncType</strong> to load text into the table cell and implement the callback <strong>TableSaveDataFuncType</strong> to save data before the field is freed.</td>
</tr>
<tr>
<td>textWithNoteTableItem</td>
<td>Editable text field and a note icon to the right of the text.</td>
<td>fontID ptr&lt;br&gt;For this item type, implement the callback function <strong>TableLoadDataFuncType</strong> to load text into the table cell and implement the callback <strong>TableSaveDataFuncType</strong> to save data before the field is freed.</td>
</tr>
</tbody>
</table>
The TablePtr type defines a pointer to a TableType.

typedef TableType *TablePtr;

You pass the table’s pointer as an argument to all table functions. You can obtain the table’s pointer using the function FrmGetObjectPtr in this way:

    tblPtr = FrmGetObjectPtr(frm, FrmGetObjectIndex(frm, tblID));

where tblID is the resource ID assigned when you created the table.

Table 20.1 TableItemType fields (continued)

<table>
<thead>
<tr>
<th>ItemType</th>
<th>Description</th>
<th>TableItemType Fields Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeTableItem</td>
<td>Not implemented.</td>
<td></td>
</tr>
<tr>
<td>narrowTextTableItem</td>
<td>Editable text with space reserved on the right side of the cell.</td>
<td>fontID, ptr, intValue</td>
</tr>
</tbody>
</table>

intValue is the number of pixels to reserve on the right side of the cell. For this item type, implement the callback function TableDrawItemFuncType to draw in the space reserved on the right side of the cell, the TableLoadDataFuncType callback function to load text into the table cell, and the callback function TableSaveDataFuncType to save data before the field is freed.
Tables
Table Data Structures

TableRowAttrType

The TableRowAttrType structure defines a row in a table.

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the TableRowAttrType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```c
typedef struct {
    UInt16 id;
    Coord height;
    UInt32 data;
    UInt16 reserved1 : 7;
    UInt16 usable : 1;
    UInt16 reserved2 : 4;
    UInt16 usable : 1;
    UInt16 reserved3;  
} TableRowAttrType;
```

Your code should treat the TableRowAttrType structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

**Field Descriptions**

- **id**
  The ID of this row. See `TblFindRowID`, `TblGetRowID`, and `TblSetRowID`.

- **height**
  Height of the row in pixels. The functions `TblSetRowHeight` and `TblGetRowHeight` set and retrieve this value.
data Any application-specific value you want to store in this row. For example, the Datebook and ToDo applications use this field to store the unique ID of the database record that is displayed in this table row. See TblFindRowData, TblGetRowData, and TblSetRowData.

reserved1 Reserved for future use.

usable If false, the row is not considered part of the current interface of the application, and it doesn’t appear on screen. Table rows have usable set to false when they are scrolled off the screen. See TblRowUsable and TblSetRowUsable. The function TblGetLastUsableRow returns the row that appears at the bottom of the screen.

masked If true and the item’s column also has a masked attribute of true, the table cell is drawn on the screen but is shaded to obscure the information that it contains. See TblSetRowMasked and TblRowMasked.

reserved2 Reserved for future use.

invalid If true, the row needs to be redrawn. See TblRowInvalid, TblMarkRowInvalid, and TblMarkTableInvalid.

staticHeight true if the row height cannot be changed, false otherwise. If false, text fields in this table row will dynamically resize to multiple lines as necessary. See TblSetRowStaticHeight.

selectable If true, the user can select individual cells in this row. See TblSetRowSelectable and TblRowSelectable.

reserved3 Reserved for future use.

Compatibility The masked field is defined only if 3.5 New Feature Set is present.
TableType
The TableType structure represents a table.

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the TableType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```c
typedef struct TableType {
    UInt16          id;
    RectangleType   bounds;
    TableAttrType   attr;
    Int16           numColumns;
    Int16           numRows;
    Int16           currentRow;
    Int16           currentColumn;
    Int16           topRow;
    TableColumnAttrType *columnAttrs;
    TableRowAttrType  *rowAttrs;
    TableItemPtr     items;
    FieldType       currentField;
} TableType;
```

Your code should treat the TableType structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

**Field Descriptions**

- **id**
  - ID value you specified when you created the table resource. This ID is included as part of the event data of `tblEnterEvent`.

- **bounds**
  - Position and size of the table object. The functions `TblGetBounds`, `FrmGetObjectBounds`, `TblSetBounds`, and `FrmSetObjectBounds` retrieve and set this value.
<table>
<thead>
<tr>
<th>attr</th>
<th>The table’s attributes. See TableAttrType.</th>
</tr>
</thead>
<tbody>
<tr>
<td>numColumns</td>
<td>Number of columns displayed by the table object. You specify the number of columns when you create the table resource, and this value cannot be changed. Access with TblGetNumberOfColumns.</td>
</tr>
<tr>
<td>numRows</td>
<td>Maximum number of visible rows in the table object. You specify this value when you create the table resource, and it does not change; however, the total number of rows in a table can change if you insert new rows in a table, and even the number of currently visible rows can change if a text field within a table cell is dynamically resized. The function TblGetNumberOfRows returns the value of this field.</td>
</tr>
<tr>
<td>currentRow</td>
<td>Row index of the currently selected table cell. Rows are numbered from top to bottom starting with 0. TblGetSelection and TblSetSelection retrieve and set the values of currentRow.</td>
</tr>
<tr>
<td>currentColumn</td>
<td>Column index of the currently selected table cell. Columns are numbered from left to right starting with 0. If the TableAttrType selected is true, then this table cell is highlighted. If selected is false, the table still considers this the “current” item, but it is not highlighted. TblGetSelection and TblSetSelection retrieve and set the values of currentColumn.</td>
</tr>
<tr>
<td>topRow</td>
<td>First visible row of the table object. Access with TblGetTopRow.</td>
</tr>
<tr>
<td>columnAttrs</td>
<td>An array of each table column’s attributes. See TableColumnAttrType.</td>
</tr>
</tbody>
</table>
Tables
Table Constants

rowAttrs An array of each row’s attributes, such as its ID, height, and whether or not it is usable, selectable, or invalid. See TableRowAttrType.

items An array of each item’s (table cell’s) attributes, such as the item type, font ID, an integer value, and a character pointer. See TableItemType.

currentField Field object the user is currently editing. The function TblGetCurrentField retrieves the value of this item.

Table Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tableDefaultColumnSpacing</td>
<td>1</td>
<td>Never used.</td>
</tr>
<tr>
<td>tableNoteIndicatorHeight</td>
<td>11</td>
<td>The height in pixels of the note indicator for tables items of type textWithNoteTableItem.</td>
</tr>
<tr>
<td>tableNoteIndicatorWidth</td>
<td>7</td>
<td>The width in pixels of the note indicator for tables items of type textWithNoteTableItem.</td>
</tr>
<tr>
<td>tableMaxTextItemSize</td>
<td>255</td>
<td>The maximum length of an editable text field within a table cell.</td>
</tr>
<tr>
<td>tblUnusableRow</td>
<td>0xffff</td>
<td>Value returned by TblGetLastUsableRow if none of the table’s rows are usable. This value is only available in version 3.5 and higher.</td>
</tr>
</tbody>
</table>

Table Resource

The Table Resource (fTBL) represents a table on screen.
Table Functions

**TblDrawTable**

**Purpose**
Draw a table.

**Declared In**
Table.h

**Prototype**
```c
void TblDrawTable (TableType *tableP)
```

**Parameters**
- `tableP` Pointer to a table object. (See `TableType`.)

**Result**
Returns nothing.

**Comments**
This function is called as part of `FrmDrawForm` when the form contains a table object.

This function draws the entire table, marking all rows valid before drawing. See the `TableItemType` struct description for more information about how each type of table cell is drawn.

When drawing cells with editable text fields (textTableItem, textWithNoteTableItem, or narrowTextTableItem), this function uses the `TableLoadDataFuncType` callback function to load the text into the table cells. The text field does not retain the text handle that your `TableLoadDataFunc` returns, meaning that you are responsible for freeing the memory that you load into the table.

When drawing `narrowTextTableItem` cells, `customTableItem` cells or `tallCustomTableItem` cells, this function uses the `TableDrawItemFuncType` callback function to draw the extra pixels after the text or to draw the entire cell.

On color systems, tables are always drawn using the same color as is used for the field background color.

When the user has set the security setting to mask private records, table cells that contain private database records are drawn as shaded cells to obscure the information they contain. You must explicitly tell the table which cells are masked by making the
appropriate calls to `TblSetRowMasked` and `TblSetColumnMasked`.

**Compatibility**  
Color support and masked private records are only supported in Palm OS® version 3.5 and higher.

In versions earlier than 3.5, this function did not erase table cells before it drew them. In earlier releases, consider calling `TblEraseTable` before calling this function, particularly if the entire table has changed, as the visual effect of drawing over a white background may be more pleasing.

**See Also**  
`TblEraseTable`, `TblRedrawTable`, `TblSetCustomDrawProcedure`  

---

**TblEditing**

**Purpose**  
Check whether a table is in edit mode.

**Declared In**  
`Table.h`

**Prototype**  
`Boolean TblEditing (const TableType *tableP)`

**Parameters**  
`-> tableP` Pointer to a table object. (See `TableType`.)

**Result**  
Returns true if the table is in edit mode, false otherwise.

**Comments**  
The table is in edit mode while the user edits a text item. More specifically, the table is in edit mode when a `tblEnterEvent` is received on an editable table cell (textTableItem, textWithNoteTableItem, or narrowTextTableItem), or when `TblGrabFocus` is called.

The table is taken out of edit mode when a the user places the pen on a note in a `textWithNoteTableItem` or when the table releases the focus (`TblReleaseFocus`).
TblEraseTable

**Purpose**
Erase a table object.

**Declared In**
Table.h

**Prototype**
void TblEraseTable (TableType *tableP)

**Parameters**
- `tableP` Pointer to a table object. (See TableType.)

**Result**
Returns nothing.

**Comments**
This function sets the table’s visible and selected attributes to false. It does not invalidate table rows.

**See Also**
TblDrawTable, TblSetCustomDrawProcedure, TblRedrawTable

TblFindRowData

**Purpose**
Return the number of the row that contains the specified data value.

**Declared In**
Table.h

**Prototype**
Boolean TblFindRowData (const TableType *tableP, UInt32 data, Int16 *rowP)

**Parameters**
- `tableP` Pointer to a table object. (See TableType.)
- `data` Row data to find.
- `rowP` Pointer to the row number (return value).

**Result**
Returns true if a match was found, false otherwise.
Comments

This function searches for a row whose attributes contain the specified data. The data is any application-specific data that you have set with TblSetRowData.

See Also

TblGetRowData, TblFindRowID, TableRowAttrType

TblFindRowID

Purpose

Return the number of the row with the specified ID.

Declared In

Table.h

Prototype

Boolean TblFindRowID (const TableType *tableP, UInt16 id, Int16 *rowP)

Parameters

- > tableP Pointer to a table object. (See TableType.)
- > id Row ID to find.
- < rowP Pointer to the row number (return value).

Result

Returns true if a match was found, false otherwise.

See Also

TblSetRowID, TblFindRowData, TableRowAttrType

TblGetBounds

Purpose

Return the bounds of a table.

Declared In

Table.h

Prototype

void TblGetBounds (const TableType *tableP, RectangleType *rP)

Parameters

- > tableP Pointer to a table object. (See TableType.)
<- rP       A RectangleType structure in which the bounds are returned.

**Result**  Returns nothing. The rP parameter contains the bounds.

**See Also**  TblGetItemBounds

---

**TblGetColumnSpacing**

**Purpose**  Return the spacing after the specified column.

**Declared In**  Table.h

**Prototype**

```c
Coord TblGetColumnSpacing (const TableType *tableP, Int16 column)
```

**Parameters**

- `-> tableP`  Pointer to a table object. (See `TableType`.)
- `-> column`  Column number (zero-based).

**Result**  Returns the spacing after column (in pixels).

This function may display a fatal error message if the column parameter is invalid.

**See Also**  TblGetColumnWidth, TblSetColumnSpacing, TblSetColumnUsable

---

**TblGetColumnWidth**

**Purpose**  Return the width of the specified column.

**Declared In**  Table.h

**Prototype**

```c
Coord TblGetColumnWidth (const TableType *tableP, Int16 column)
```

**Parameters**

- `-> tableP`  Pointer to a table object. (See `TableType`.)
Table Functions

> column Column number (zero-based).

**Result** Returns the width of a column (in pixels). This function may display a fatal error message if the column parameter is invalid.

**See Also** TblGetColumnSpacing, TblSetColumnWidth, TblSetColumnUsable

### TblGetCurrentField

**Purpose** Return a pointer to the FieldType in which the user is currently editing a text item.

**Declared In** Table.h

**Prototype** FieldPtr TblGetCurrentField (const TableType *tableP)

**Parameters**

- > tableP Pointer to a table object. (See TableType.)

**Result** Returns a pointer to the currently selected field, or NULL if the table is not in edit mode.

**See Also** TblGetSelection

### TblGetItemBounds

**Purpose** Return the bounds of an item in a table.

**Declared In** Table.h

**Prototype** void TblGetItemBounds (const TableType *tableP, Int16 row, Int16 column, RectangleType *rP)

**Parameters**

- > tableP Pointer to a table object. (See TableType.)
- > row Row number of the item (zero-based).
- > column Column number of the item (zero-based).
<- rP Pointer to a structure that holds the bounds of the item.

Result Returns nothing. Stores the bounds in rP. This function may raise a fatal exception if the row or column parameter specifies a row or column that does not appear on screen.

**TblGetItemFont**

**Purpose** Return the font used to display a table item.

**Declared In** Table.h

**Prototype** FontID TblGetItemFont (const TableType *tableP, Int16 row, Int16 column)

**Parameters**
- > tableP Pointer to a table object. (See TableType.)
- > row Row number of the item (zero-based).
- > column Column number of the item (zero-based).

**Result** Returns the ID of the font used for the table item at the row and column indicated. This function may display a fatal error message if the row or column parameter specifies a row or column that is not on the screen.

**Comments** This function returns the value stored in the fontID field for this table item. Only certain types of table items use the font specified by the fontID field when they are displayed. The TableItemType description specifies what font is used to display each type of table item.

**Compatibility** Implemented only if 3.0 New Feature Set is present.

**See Also** TblSetItemFont
### TblGetItemInt

**Purpose**  
Return the integer value stored in a table item.

**Declared In**  
Table.h

**Prototype**  
```c
Int16 TblGetItemInt (const TableType *tableP, Int16 row, Int16 column)
```

**Parameters**
- `-> tableP`  
  Pointer to a table object. (See `TableType`.)
- `-> row`  
  Row number of the item (zero-based).
- `-> column`  
  Column number of the item (zero-based).

**Result**  
Returns the integer value. This function may display a fatal message if the `row` or `column` does not appear on the screen.

**Comments**  
This function returns the value stored in the `intValue` field for this table item. Certain types of table items display the value stored in `intValue`, and other types display the value pointed to by the `ptr` field. See the `TableItemType` description for details. If the `intValue` was never set for this table item, this function returns 0.

**See Also**  
TblSetItemInt, TblGetItemPtr

### TblGetItemPtr

**Purpose**  
Return the pointer value stored in a table item

**Declared In**  
Table.h

**Prototype**  
```c
void *TblGetItemPtr (const TableType *tableP, Int16 row, Int16 column)
```

**Parameters**
- `-> tableP`  
  Pointer to a table object. (See `TableType`.)
- `-> row`  
  Row number of the item (zero-based).
Column number of the item (zero-based).

**Result**  
Returns the item’s pointer value or `NULL` if the item does not have a pointer value. This function may display a fatal message if the `row` or `column` parameter is invalid.

**Comments**  
This function returns the value stored in the `ptr` field for this table item. Certain types of table items display the value pointed to by the `ptr`, and other types display the value stored in the `intValue` field. See the `TableItemType` description for details. An application may have set the value of the `ptr` field anyway, even for items that use the `intValue`. This function always returns that value.

**Compatibility**  
Implemented only if 3.5 New Feature Set is present. In earlier versions, you can implement this function using the following code:

```c
return tableP->items[row * tableP->numColumns + column].ptr;
```

**See Also**  
TblSetItemPtr

---

**TblGetLastUsableRow**

**Purpose**  
Return the last row in a table that is usable (visible).

**Declared In**  
Table.h

**Prototype**  
`Int16 TblGetLastUsableRow(const TableType *tableP)`

**Parameters**  
`-> tableP`  
Pointer to a table object. (See `TableType`.)

**Result**  
Returns the row index (zero-based) or `tblUnusableRow` if there are no usable rows.

**See Also**  
TblGetRowData, TblGetRowID
TblGetNumberOfColumns

**Purpose**
Return the number of columns in a table.

**Declared In**
Table.h

**Prototype**
Int16 TblGetNumberOfColumns
(const TableType *tableP)

**Parameters**
-> tableP Pointer to a TableType.

**Result**
This function returns the number of columns in a table.

**Compatibility**
Implemented only if 4.0 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TblGlueGetNumberOfColumns. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**  
TblGetTopRow, TblSetSelection

TblGetNumberOfRows

**Purpose**
Return the number of rows in a table.

**Declared In**
Table.h

**Prototype**
Int16 TblGetNumberOfRows
(const TableType *tableP)

**Parameters**
-> tableP Pointer to a table object. (See TableType.)

**Result**
Returns the maximum number of visible rows in the specified table.

**Comments**
Note that even though you can add and remove rows to and from a table, the value returned by this function does not change. The value returned by this function indicates the maximum number of
rows that can be displayed on the screen at one time. It is set when you create the table resource.

**TblGetRowData**

**Purpose**

Return the data value of the specified row.

**Declared In**

Table.h

**Prototype**

UInt32 TblGetRowData (const TableType *tableP, Int16 row)

**Parameters**

- `tableP` Pointer to a table object. (See TableType.)
- `row` Number of the row (zero-based).

**Result**

Returns the application-specific data stored for this row, if any. Returns 0 if there is no application-specific data value.

This function may display a fatal error message if the row parameter is invalid.

**See Also**

TblFindRowData, TblSetRowData, TableRowAttrType

**TblGetRowHeight**

**Purpose**

Return the height of the specified row.

**Declared In**

Table.h

**Prototype**

Coord TblGetRowHeight (const TableType *tableP, Int16 row)

**Parameters**

- `tableP` Pointer to a table object. (See TableType.)
Tables
Table Functions

>> row Number of the row (zero-based).

Result Returns the height in pixels. This function may display a fatal error message if the row parameter is invalid.

See Also TblGetItemBounds, TblSetRowHeight

TblGetRowID

Purpose Return the ID value of the specified row.

Declared In Table.h

Prototype UInt16 TblGetRowID (const TableType *tableP, Int16 row)

Parameters -> tableP Pointer to a table object. (See TableType.)
-> row Number of the row (zero-based).

Result Returns the ID value of the row in the table.
This function may display a fatal error message if the row parameter is invalid.

See Also TblGetRowData, TblSetRowID, TblFindRowID, TableRowAttrType

TblGetSelection

Purpose Return the row and column of the currently selected table item.

Declared In Table.h

Prototype Boolean TblGetSelection (const TableType *tableP, Int16 *rowP, Int16 *columnP)

Parameters -> tableP Pointer to a table object. (See TableType.)
<-rowP, columnP
The row and column indexes (zero-based) of the currently selected item.

Result
Returns true if the item is highlighted, false if not.

See Also
TblSetRowSelectable

TblGetTopRow

Purpose
Return the top row visible row of a table.

Declared In
Table.h

Prototype
Int16 TblGetTopRow(const TableType *tableP)

Parameters
-> tableP Pointer to a TableType.

Result
This function returns the top visible row in a table.

Compatibility
Implemented only if 4.0 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TblGlueGetToRow. For more information, see Chapter 75, “PalmOSGlue Library.”

See Also
TblGetNumberOfColumns, TblSetSelection

TblGrabFocus

Purpose
Put a table into edit mode.

Declared In
Table.h

Prototype
void TblGrabFocus (TableType *tableP, Int16 row, Int16 column)

Parameters
-> tableP Pointer to a table object. (See TableType.)
- row: Current row to be edited (zero-based).
- column: Current column to be edited (zero-based).

**Result**

Returns nothing. This function may display a fatal error message if the table already has the focus or if the row or column parameter is invalid.

**Comments**

This function puts the table into edit mode and sets the current item to the one at the row and column passed in. An editable field must exist in the coordinates passed to this function.

You must call **FrmSetFocus** before calling this function. **FrmSetFocus** releases the focus from the object that previously had it and sets the form’s internal structures. After calling this function, you must call **FldGrabFocus** to display the insertion point in the field. (You can use **TblGetCurrentField** to obtain a pointer to the field.)

For example, the following function from the Address Book application sets the focus in an editable field within a table:

```c
static void EditViewRestoreEditState () {
    Int16      row;
    FormPtr    frm;
    TablePtr   table;
    FieldPtr   fld;

    if (CurrentFieldIndex == noFieldIndex)
        return;

    // Find the row that the current field is in.
    table = GetObjectPtr (EditTable);
    if ( ! TblFindRowID (table,
                      CurrentFieldIndex, &row) )
        return;

    frm = FrmGetActiveForm ();
    FrmSetFocus (frm, FrmGetObjectIndex (frm, EditTable));
    TblGrabFocus (table, row, editDataColumn);
}```
// Restore the insertion point position.
    fld = TblGetCurrentField (table);
    FldSetInsPtPosition (fld, EditFieldPosition);
    FldGrabFocus (fld);
}

See Also  TblReleaseFocus

TblHandleEvent

Purpose  Handle an event for the table.

Declared In  Table.h

Prototype  Boolean TblHandleEvent (TableType *tableP, EventType *event)

Parameters
  -> tableP          Pointer to a table object. (See TableType.)
  -> event          The event to be handled.

Result
  Returns true if the event was handled, false if it was not.

Comments
  Returns false if the table is not an editable table.
  If the table is editable, this function passes along any
  keyDownEvent, fldEnterEvent, or menuCmdBarOpenEvent to
  the currently selected field.

  When a fldHeightChangedEvent occurs, this function changes
  the height of the specified field as indicated by the event. If the field
  being resized is going to scroll off the bottom of the screen, then
  instead the table scrolls the rows above it up off the top. Otherwise,
  the table is scrolled downward and rows below the current row are
  scrolled off the bottom as necessary.

  Note that the fldHeightChangedEvent is only handled for
  dynamically sized fields. See the descriptions of FieldAttrType
  and FldMakeFullyVisible for more information.

  When a penDownEvent occurs, the table checks to see if the focus is
  being changed. If it is and the user was previously editing a text
field within the table, it saves the data in the table cell using the `TableSaveDataFuncType` callback function, then it enqueues a `tblEnterEvent` with the new row and column that are selected.

When a `tblEnterEvent` occurs, this function tracks the pen until it is lifted. If the pen is lifted within the bounds of the same item it went down in, a `tblSelectEvent` is added to the event queue; if not, a `tblExitEvent` is added to the event queue.

**Table HasScrollBar**

**Purpose**  
Set the `hasScrollBar` attribute in the table. (See `TableAttrType`.)

**Declared In**  
`Table.h`

**Prototype**  
```c
void TblHasScrollBar (TableType *tableP, Boolean hasScrollBar)
```

**Parameters**  
- `tableP`  
  Pointer to a table object. (See `TableType`.)
- `hasScrollBar`  
  true to set the attribute, false to unset it.

**Result**  
Returns nothing.

**Comments**  
Your application must scroll the table itself and keep track of the scroll values. See the `ListViewUpdateScrollers` function in the Memo example application (`MemoMain.c`) for an example of setting scroll bar values for a table.

**Compatibility**  
Implemented only if 2.0 New Feature Set is present.
## TblInsertRow

### Purpose
Insert a row into the table before the specified row.

### Declared In
Table.h

### Prototype
void TblInsertRow (TableType *tableP, Int16 row)

### Parameters
- `tableP`: Pointer to a table object. (See TableType.)
- `row`: Row to insert (zero-based).

### Result
Returns nothing.

### Comments
The number of rows in a table is the maximum number of rows displayed on the screen. Unlike a multi-line text field, there is no notion of a table that is bigger than the available screen. For this reason, this function does not increase the number of table rows.

Instead of keeping track of a total number of rows in the table and a number of rows displayed on the screen, tables mark any row that isn’t currently displayed with a usable value of `false`. (See TableRowAttrType.)

The newly inserted row is marked as invalid, unusable, and not masked. If you want to display the newly inserted row, call TblSetRowUsable after making sure that the row displays a value and then call TblRedrawTable when you are ready to draw the table.

### See Also
TblRemoveRow, TblSetRowUsable, TblSetRowSelectable
TblMarkRowInvalid

**Purpose**
Mark the row invalid.

**Declared In**
Table.h

**Prototype**
```c
void TblMarkRowInvalid (TableType *tableP, Int16 row)
```

**Parameters**
- `tableP` Pointer to a table object. (See TableType.)
- `row` Row number (zero-based).

**Result**
Returns nothing.

**Comments**
After calling this function, call TblRedrawTable to redraw all rows marked invalid.
This function may display a fatal error message if the row parameter is invalid.

**See Also**
TblRemoveRow, TblSetRowUsable, TblSetRowSelectable, TblMarkTableInvalid, TblRowInvalid, TableRowAttrType

TblMarkTableInvalid

**Purpose**
Mark all the rows in a table invalid.

**Declared In**
Table.h

**Prototype**
```c
void TblMarkTableInvalid (TableType *tableP)
```

**Parameters**
- `tableP` Pointer to a table object. (See TableType.)

**Result**
Returns nothing.
After calling this function, you must call `TblRedrawTable` to redraw all rows.

**See Also**
- `TblEraseTable`
- `TblRedrawTable`
- `TableRowAttrType`

### TblRedrawTable

**Purpose**
Redraw the rows of the table that are marked invalid.

**Declared In**
`Table.h`

**Prototype**
```c
void TblRedrawTable (TableType *tableP)
```

**Parameters**
- `tableP` Pointer to a table object. (See `TableType`.)

**Result**
Returns nothing.

**Comments**
This function draws the invalid rows in the table. See the `TableItemType` struct description for more information about how each type of table cell is drawn.

When drawing cells with editable text fields (textTableItem, textWithNoteTableItem, or narrowTextTableItem), this function uses the `TableLoadDataFuncType` callback function to load the text into the table cells. The text field does not retain the text handle that your `TableLoadDataFunc` returns, meaning that you are responsible for freeing the memory that you load into the table.

When drawing narrowTextTableItem cells, customTableItem cells, or tallCustomTableItem cells, this function uses the `TableDrawItemFuncType` callback function to draw the extra pixels after the text or to draw the entire cell.

On color systems, tables are always drawn using the same color as is used for the field background color.

When the user has set the security setting to mask private records, table cells that contain private database records are drawn as shaded cells to obscure the information they contain. You must explicitly tell the table which cells are masked by making the
appropriate calls to `TblSetRowMasked` and `TblSetColumnMasked`.

**Compatibility**
Color support and masked private records are only supported in Palm OS version 3.5 and higher.

**See Also** `TblMarkTableInvalid`, `TblMarkRowInvalid`, `TblDrawTable`

---

**TblReleaseFocus**

**Purpose**
Release the focus.

**Declared In**
`Table.h`

**Prototype**
```c
void TblReleaseFocus (TableType *tableP)
```

**Parameters**
- `-> tableP` Pointer to a table object.

**Result**
Returns nothing.

**Comments**
You typically do not call this function yourself. Instead, call `FrmSetFocus` with an object index of `noFocus` to notify the form that the table has lost focus. The form code calls `TblReleaseFocus` for you.

If the current item is a text item, the `TableSaveDataFuncType` callback function is called to save the text in the currently selected field, the memory allocated for editing is released, and the insertion point is turned off.

Also note that you might have to call `FldReleaseFocus` if the focus is in an editable text field and that field uses a custom drawing function (`TableDrawItemFuncType`).

**See Also** `TblGrabFocus`
**TblRemoveRow**

**Purpose**
Remove the specified row from the table.

**Declared In**
Table.h

**Prototype**
void TblRemoveRow (TableType *tableP, Int16 row)

**Parameters**
- `tableP` Pointer to a table object. (See TableType.)
- `row` Row to remove (zero-based).

**Result**
Returns nothing. This function may raise a fatal error message if an invalid row is specified.

**Comments**
The number of rows in the table is not decreased; instead, this row is moved from its current spot to the end of the table and is marked unusable so that it won’t be displayed when the table is redrawn. This function does not visually update the display. To update the display, call TblRedrawTable.

**See Also**
TblInsertRow, TblSetRowUsable, TblSetRowSelectable, TblMarkRowInvalid

**TblRowInvalid**

**Purpose**
Return whether a row is invalid.

**Declared In**
Table.h

**Prototype**
Boolean TblRowInvalid (const TableType *tableP, Int16 row)

**Parameters**
- `tableP` Pointer to a table object. (See TableType.)
- `row` Row number (zero-based).

**Result**
Returns true if the row is invalid, false if it’s valid. This function may raise a fatal error message if the `row` parameter is invalid.
Comments
Invalid rows need to be redrawn. Use TblRedrawTable to do so.

See Also
TblMarkRowInvalid, TblMarkTableInvalid

**TblRowMasked**

**Purpose**
Return whether a row is masked.

**Declared In**
Table.h

**Prototype**
Boolean TblRowMasked (const TableType *tableP, Int16 row)

**Parameters**
- `tableP`  
  Pointer to a table object. (See TableType.)
- `row`  
  Row number (zero-based).

**Result**
Returns true if the row is masked, false otherwise.

**Comments**
Your code should set a row to masked if it contains a private database record and the user has set the display preference for private records to masked. Masked cells are displayed as shaded. 

Note that a table cell is not masked unless both its row and column are masked. This allows non-private information in the row item to remain visible. For example, the Datebook application shows the time for a private appointment, but it does not show the description.

**Compatibility**
Implemented only if 3.5 New Feature Set is present.

**See Also**
TblSetRowMasked, TblSetColumnMasked, TableRowAttrType, SecSelectViewStatus
**TblRowSelectable**

**Purpose**
Return whether the specified row is selectable.

**Declared In**
Table.h

**Prototype**
```c
Boolean TblRowSelectable (const TableType *tableP, Int16 row)
```

**Parameters**
- `tableP`  
  Pointer to a table object. (See `TableType`.)
- `row`  
  Row number (zero-based).

**Result**
Returns `true` if the row is selectable, `false` if it’s not.

**Comments**
Rows that are not selectable don’t highlight when touched.

**See Also**
`TableRowAttrType`

---

**TblRowUsable**

**Purpose**
Determine whether the specified row is usable.

**Declared In**
Table.h

**Prototype**
```c
Boolean TblRowUsable (const TableType *tableP, Int16 row)
```

**Parameters**
- `tableP`  
  Pointer to a table object. (See `TableType`.)
- `row`  
  Row number (zero-based).

**Result**
Returns `true` if the row is usable, `false` if it’s not.

This function may display a fatal error message if the `row` parameter is invalid.
### TblSelectItem

#### Purpose
Select (highlight) the specified item. If there is already a selected item, it is unhighlighted.

#### Declared In
`Table.h`

#### Prototype
```c
void TblSelectItem (TableType *tableP, Int16 row, Int16 column)
```

#### Parameters
- `-> tableP` Pointer to a table object. (See `TableType`.)
- `-> row` Row of the item to select (zero-based).
- `-> column` Column of the item to select (zero-based).

#### Result
Returns nothing.

This function may display a fatal error message if the `column` or `row` parameter point to an item that is not on the screen.

#### Comments
If `row` contains a masked private database record, then the item remains unselected.

This function cannot highlight an entire row; it can only highlight one cell in a row, and it always unhighlights the previously selected table cell. If you want to select an entire row, either create a table that has a single column, or write your own selection code.

If the selected item is a multi-line text field or a text field with a nonstandard height, this function only highlights the top eleven pixels. If you want a larger area highlighted, you must write your own selection code.

#### See Also
- `TblRowSelectable`, `TblGetLastUsableRow`, `TblSetRowUsable`
- `TblGetItemBounds`, `TblGetItemInt`
**TblSetBounds**

**Purpose**
Sets the bounds of a table.

**Declared In**
Table.h

**Prototype**
void TblSetBounds (TableType *tableP, const RectangleType *rP)

**Parameters**
- `tableP` Pointer to a table object. (See `TableType`.)
- `rP` Pointer to a `RectangleType` structure that specifies the bounds for the table.

**Result**
Returns nothing.

**Compatibility**
Implemented only if 2.0 New Feature Set is present.

**TblSetColumnEditIndicator**

**Purpose**
Set the column attribute that controls whether a column highlights when the table is in edit mode.

**Declared In**
Table.h

**Prototype**
void TblSetColumnEditIndicator (TableType *tableP, Int16 column, Boolean editIndicator)

**Parameters**
- `tableP` Pointer to a table object. (See `TableType`.)
- `column` Column number (zero based).
- `editIndicator` true to highlight, false to turn off highlight.

**Result**
Returns nothing.

**Comments**
The edit indicator controls whether the item in the column is highlighted when it is selected. By default, text field items have the
editIndicator value of false, while all other table item types have an edit indicator of true.

When the table is drawn, only the leftmost contiguous set of items with the edit indicator set are drawn as highlighted. That is, if columns 1, 2, and 4 have an edit indicator of true and column 3 has an edit indicator of false, only the items in column 1 and 2 are drawn as highlighted when selected. Column 4 items are not drawn as highlighted.

Compatibility
Implemented only if 2.0 New Feature Set is present.

See Also
TableColumnAttrType

TblSetColumnMasked

Purpose
Set whether the column is masked.

Declared In
Table.h

Prototype
void TblSetColumnMasked (TableType *tableP, Int16 column, Boolean masked)

Parameters
-> tableP Pointer to a table object. (See TableType.)
-> column Column number (zero-based).
-> masked true to have the column be masked, false otherwise.

Result
Returns nothing.

Comments
Masked cells are displayed as shaded. You should set a column to masked if its contents should be hidden when it contains information from a private database record and the user has set the display preference for private records to masked.

A table cell is not masked unless both its row and column are masked. This allows non-private information in the row item to remain visible. For example, the Datebook application shows the time for a private appointment, but it does not show the description.
Because the number of columns is static, you only need to call this function once per column when you first set up the table. The masked attribute on the row will determine if the contents of the table cell are actually displayed as shaded.

### Compatibility
Implemented only if [3.5 New Feature Set](#) if present.

### See Also
TblRowMasked, TblSetRowMasked, TableColumnAttrType, SecSelectViewStatus

### TblSetColumnSpacing

#### Purpose
Set the spacing after the specified column.

#### Declared In
Table.h

#### Prototype
```c
void TblSetColumnSpacing (TableType *tableP, Int16 column, Coord spacing)
```

#### Parameters
- **tableP** Pointer to a table object. (See `TableType`.)
- **column** Column number (zero-based).
- **spacing** Spacing after the column in pixels.

#### Result
Returns nothing.

This function may display a fatal error message if the `column` parameter is invalid.

#### See Also
TblSetColumnUsable, TableColumnAttrType
**TblSetColumnUsable**

**Purpose**
Set a column in a table to usable or unusable.

**Declared In**
Table.h

**Prototype**
```c
void TblSetColumnUsable (TableType *tableP,
Int16 column, Boolean usable)
```

**Parameters**
- `-> tableP` Pointer to a table object. (See `TableType`.)
- `-> column` Column number (zero-based).
- `-> usable` true for usable or false for not usable.

**Result**
Returns nothing.

This function may display a fatal error message if the `column` parameter is invalid.

**Comments**
Columns that are not usable do not display.

**See Also**
`TblMarkRowInvalid`, `TableColumnAttrType`

**TblSetColumnWidth**

**Purpose**
Set the width of the specified column.

**Declared In**
Table.h

**Prototype**
```c
void TblSetColumnWidth (TableType *tableP,
Int16 column, Coord width)
```

**Parameters**
- `-> tableP` Pointer to a table object. (See `TableType`.)
- `-> column` Column number (zero-based).
- `-> width` Width of the column (in pixels).

**Result**
Returns nothing.
This function may display a fatal error message if the column parameter is invalid.

See Also  TblGetColumnWidth, TableColumnAttrType

**TblSetCustomDrawProcedure**

**Purpose**  Set the custom draw callback procedure for the specified column.

**Declared In**  Table.h

**Prototype**  

```
void TblSetCustomDrawProcedure
  (TableType *tableP, Int16 column,
   TableDrawItemFuncPtr drawCallback)
```

**Parameters**  

- `-> tableP`  Pointer to a table object. (See TableType.)
- `-> column`  Column number.
- `-> drawCallback`  Callback function.

**Result**  Returns nothing.

**Comments**  

The custom draw callback function is used to draw table items with a TableItemStyleType of customTableItem or tallCustomTableItem. See the TableItemType description for more information.

This function may display a fatal error message if the column parameter is invalid.

See Also  TableDrawItemFuncType, TblDrawTable, TableColumnAttrType
**TblSetItemFont**

**Purpose**  
Set the font used to display a table item.

**Declared In**  
Table.h

**Prototype**  
```c
void TblSetItemFont (TableType *tableP, 
                   Int16 row, Int16 column, FontID fontID)
```

**Parameters**
- `tableP`  
  Pointer to a table object. (See `TableType`.)
- `row`  
  Row number of the item (zero-based).
- `column`  
  Column number of the item (zero-based).
- `fontID`  
  ID of the font to be used.

**Result**  
Returns nothing.

**Comments**  
This function sets the value stored in the `fontID` field for this table item. Only certain types of table items use the font specified by the `fontID` field when they are displayed. The `TableItemType` description specifies what font is used to display each type of table item. It is not an error to set the `fontID` for a table item that does not use it.

This function may display a fatal error message if the `row` or `column` parameter specifies a row or column that is not on the screen.

**Compatibility**  
Implemented only if 3.0 New Feature Set is present.

**See Also**  
`TblGetItemFont`
**TblSetItemInt**

**Purpose**  
Set the integer value of the specified item.

**Declared In**  
Table.h

**Prototype**  
```c
void TblSetItemInt (TableType *tableP, Int16 row, Int16 column, Int16 value)
```

**Parameters**
- `-> tableP`  
  Pointer to a table object. (See `TableType`.)
- `-> row`  
  Row number of the item (zero-based).
- `-> column`  
  Column number of the item (zero-based).
- `-> value`  
  Any byte value (an integer).

**Result**  
Returns nothing.

This function may display a fatal error message if the `row` or `column` parameter is invalid.

**Comments**  
You typically use this function when setting up and initializing a table for the first time to set the value of each table cell.

This function sets the value stored in the intValue field for this table item. Certain types of table items display the value stored in `intValue`, and other types display the value pointed to by the ptr field. See the `TableItemType` description for details. If you set the `intValue` of an item that displays its ptr value, it is not an error. An application can store whatever value it wants in the `intValue` field; however, be aware that this has nothing to do with the value displayed by such a table cell.

**See Also**  
`TblGetItemInt`, `TblSetItemPtr`
TblSetItemPtr

Purpose
Set the item to the specified pointer value.

Declared In
Table.h

Prototype
void TblSetItemPtr (TableType *tableP, Int16 row, Int16 column, void *value)

Parameters
- tableP Pointer to a table object. (See TableType.)
- row Row number of the item (zero-based).
- column Column number of the item (zero-based).
- value Pointer to data to display in the table item.

Result
Returns nothing.

This function may display a fatal error message if the row or column parameter is invalid.

Comments
This function sets the value stored in the ptr field for this table item. Certain types of table items display the value pointed to by ptr, and other types display the value stored in the intValue field. See the TableItemType description for details. If you set the ptr of an item that displays its intValue, it is not an error. An application can store whatever value it wants in the ptr field; however, be aware that this has nothing to do with the value displayed by such a table cell.

See Also
TblGetItemPtr, TblSetItemInt
**TblSetItemStyle**

**Purpose**  
Set the type of item to display; for example, text, numbers, dates, and so on.

**Declared In**  
Table.h

**Prototype**  
void TblSetItemStyle (TableType *tableP, Int16 row, Int16 column, TableItemStyleType type)
Parameters

- `tableP` Pointer to a table object. (See `TableType`.)
- `row` Row number of the item (zero-based).
- `column` Column number of the item (zero-based).
- `type` The type of item, such as an editable text field or a check box. See `TableItemType` for a list of possible values.

Result

Returns nothing.

This function may display a fatal error message if the `row` or `column` parameter is invalid.

Comments

You typically use this function when first setting up and initializing a table; you do not dynamically change item styles.

Follow this function with a call to either `TblSetItemInt` or `TblSetItemPtr` to set the value displayed by the table item. You should call one or the other of these functions depending on the type of table item you specified. See the table in the `TableItemType` description for details.

Note also that a table column always contains items of the same type. For example, you might initialize a table using this code:

```c
for (row = 0; row < rowsInTable; row++) {
    TblSetItemStyle (table, row, completedColumn, checkboxTableItem);
    TblSetItemStyle (table, row, priorityColumn, numericTableItem);
    TblSetItemStyle (table, row, descColumn, textTableItem);
    TblSetItemStyle (table, row, dueDateColumn, customTableItem);
    TblSetItemStyle (table, row, categoryColumn, customTableItem);
}
```

See Also

`TblSetCustomDrawProcedure`
**TblSetLoadDataProcedure**

**Purpose**
Set the load-data callback procedure for the specified column.

**Declared In**
Table.h

**Prototype**

```c
void TblSetLoadDataProcedure (TableType *tableP, Int16 column, 
TableLoadDataFuncPtr loadDataCallback)
```

**Parameters**

- `-> tableP` Pointer to a table object. (See `TableType`.)
- `-> column` Column number (zero-based).
- `-> loadDataCallback` Callback procedure. See `TableLoadDataFuncType`.

**Result**
Returns nothing.

**Comments**
The callback procedure is used to load the data values of a table item. See the `TableLoadDataFuncType` for more information on writing the callback function.

You typically use this function when first setting up and initializing a table.

**See Also**
`TblSetCustomDrawProcedure`

**TblSetRowData**

**Purpose**
Set the data value of the specified row. The data value is a placeholder for application-specific values.

**Declared In**
Table.h

**Prototype**

```c
void TblSetRowData (TableType *tableP, Int16 row, 
UInt32 data)
```

**Parameters**

- `-> tableP` Pointer to a table object. (See `TableType`.)
### TblSetRowData

**Purpose**
Set the data of the specified row.

**Declared In**
Table.h

**Prototype**
```c
void TblSetRowData (TableType *tableP, Int16 row, void *data);
```

**Parameters**
- `-> tableP` Pointer to a table object. (See `TableType`.)
- `-> row` Row number (zero-based).
- `-> data` Application-specific data value to store for this row. For example, the Datebook and ToDo applications use this field to store the unique ID of the database record displayed by this row.

**Result**
Returns nothing.

This function may display a fatal error message if the `row` parameter is invalid.

**See Also**
TblGetRowData, TblFindRowData

### TblSetRowHeight

**Purpose**
Set the height of the specified row.

**Declared In**
Table.h

**Prototype**
```c
void TblSetRowHeight (TableType *tableP, Int16 row, Coord height);
```

**Parameters**
- `-> tableP` Pointer to a table object. (See `TableType`.)
- `-> row` Row number (zero-based).
- `-> height` New height in pixels.

**Result**
Returns nothing.

This function may display a fatal error message if the `row` parameter is invalid.

**See Also**
TblGetRowHeight, TblSetRowStaticHeight
TblSetRowID

**Purpose**  Set the ID value of the specified row.

**Declared In**  Table.h

**Prototype**  
void TblSetRowID (TableType *tableP, Int16 row, UInt16 id)
**Tables**

*Table Functions*

**TblSetRowMasked**

**Purpose**
Set a row in a table to masked or unmasked.

**Declared In**
Table.h

**Prototype**
```c
void TblSetRowMasked (TableType *tableP, Int16 row, Boolean masked)
```

**Parameters**
- `-> tableP` Pointer to a table object. (See `TableType`.)
- `-> row` Row number (zero-based).
- `-> masked` True to have the row be masked, false otherwise.

**Result**
Returns nothing.

**Comments**
Masked cells are displayed as shaded. You should call this function before drawing or redrawing the table. If a table row contains a private database record and the user has set the display preference for private records to masked, you must call this function on that row. For example:

```c
UInt16 attr;
privateRecordViewEnum privateRecordStatus;
Boolean masked;

privateRecordStatus = (privateRecordViewEnum)
PrefGetPreference(prefShowPrivateRecords);
....
DmRecordInfo (ToDoDB, recordNum, &attr, NULL, NULL);
masked = ((attr & dmRecAttrSecret) &&
   (privateRecordStatus == maskPrivateRecords));
TblSetRowMasked(tableP, row, masked);

Note that a table cell is not masked unless both its row and column
are masked. This allows non-private information in the row item to
remain visible. For example, the Datebook application shows the
time for a private appointment, but it does not show the description.

**Compatibility**
Implemented only if 3.5 New Feature Set is present.

**See Also**
TblRowMasked, TblSetColumnMasked, TableRowAttrType,
SecSelectViewStatus

---

**TblSetRowSelectable**

**Purpose**
Set a row in a table to selectable or nonselectable.

**Declared In**
Table.h

**Prototype**
void TblSetRowSelectable (TableType *tableP,
Int16 row, Boolean selectable)

**Parameters**
-> tableP Pointer to a table object. (See TableType.)
-> row Row number (zero-based).
-> selectable true or false.

**Result**
Returns nothing.

This function may display a fatal error message if the row
parameter is invalid.

**Comments**
Rows that are not selectable don’t highlight when touched.

**See Also**
TblRowSelectable, TblSetRowUsable, TableRowAttrType
TblSetRowStaticHeight

Purpose  Set the static height attribute of a row.

Declared In  Table.h

Prototype  void TblSetRowStaticHeight (TableType *tableP, Int16 row, Boolean staticHeight)

Parameters  
- > tableP  Pointer to a table object. (See TableType.)
- > row  Row number (zero-based).
- > staticHeight  true to set the static height, false to unset it.

Result  Nothing.

This function may display a fatal error message if the row parameter is invalid.

Comments  A row that has its static height attribute set will not expand or contract the height of the row as text is added or removed from a text item.

Compatibility  Implemented only if 2.0 New Feature Set is present.

TblSetRowUsable

Purpose  Set a row in a table to usable or unusable. Rows that are not usable do not display.

Declared In  Table.h

Prototype  void TblSetRowUsable (TableType *tableP, Int16 row, Boolean usable)

Parameters  
- > tableP  Pointer to a table object. (See TableType.)
- > row  Row number (zero-based).
- usable 

**Result**

Returns nothing.

This function may display a fatal error message if the row parameter is invalid.

**See Also**  
TblRowUsable, TblSetRowSelectable

## TblSetSaveDataProcedure

**Purpose**

Set the save-data callback procedure for the specified column.

**Declared In**  
Table.h

**Prototype**

```c
void TblSetSaveDataProcedure (TableType *tableP, Int16 column, TableSaveDataFuncPtr saveDataCallback)
```

**Parameters**

- `tableP` Pointer to a table object. (See TableType.)
- `column` Column number (zero-based).
- `saveDataCallback` Callback function. See TableSaveDataFuncType.

**Result**

Returns nothing.

This function may display a fatal error message if the column parameter is invalid.

**Comments**

The callback procedure is called when the table object determines the data of a text object needs to be saved.

**See Also**  
TblSetCustomDrawProcedure
**TblSetSelection**

**Purpose**
Set a table item.

**Declared In**
Table.h

**Prototype**
void TblSetSelection (TableType *tableP, Int16 row, Int16 column)

**Parameters**
- tableP: Pointer to a TableType.
- row: Table row.
- column: Table column.

**Result**
Returns nothing.

**Comments**
This function sets a table item, determined by the row and column arguments, as the current selection. TblDrawTable must be called afterwards to update the UI.

**Compatibility**
Implemented only if 4.0 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TblGlueSetSelection. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**
TblGetNumberOfColumns, TblGetTopRow
TblUnhighlightSelection

**Purpose**  Unhighlight the currently selected item in a table.

**Declared In**  Table.h

**Prototype**  void TblUnhighlightSelection (TableType *tableP)

**Parameters**  
- `-> tableP`  Pointer to a table object. (See **TableType**.)

**Result**  Returns nothing.

Application-Defined Functions

TableDrawItemFuncType

**Purpose**  Draw a custom table item.

**Declared In**  Table.h

**Prototype**  void TableDrawItemFuncType (void *tableP, Int16 row, Int16 column, RectangleType *bounds)

**Parameters**  
- `-> tableP`  Pointer to a table object. (See **TableType**.)
- `-> row`  Row number of the item to be drawn (zero-based).
- `-> column`  Column number of the item to be drawn (zero-based).
- `-> bounds`  The area of the screen in which the item is to be drawn.

**Result**  Returns nothing.

**Comments**  This function is called during **TblDrawTable** and **TblRedrawTable**.
Tables
Application-Defined Functions

You implement a custom drawing function when your table contains items of type customTableItem or tallCustomTableItem (to draw the entire item) or narrowTextTableItem (to draw whatever is required in the space between the text and the right edge of the table cell).

You may implement a custom drawing function to include any style of information in the table. If you don’t like the way a predefined item is drawn, you may prefer to use a customTableItem or tallCustomTableItem instead. For example, if you want to include a date in your table but you want it to show the year as well as the month and day, you should implement a custom drawing function.

See Also  TblSetCustomDrawProcedure, TableItemType

TableLoadDataFuncType

Purpose  Load data into a column.

Declared In  Table.h

Prototype  Err TableLoadDataFuncType (void *tableP, Int16 row, Int16 column, Boolean editable, MemHandle *dataH, Int16 *dataOffset, Int16 *dataSize, FieldPtr fld)

Parameters

- -> tableP  Pointer to a table object. (See TableType.)
- -> row  Row number of the table item to load.
- -> column  Column number of the table item to load.
- -> editable  If true, the table is currently being edited. If false, the table is being drawn but not necessarily being edited.
- <- dataH  Unlocked handle of a block containing a null-terminated text string.
- <- dataOffset  Offset from start of block to start of the text string.
<- dataSize  Allocated size of text string, not the string length.
-> fld     Pointer to the text field in this table cell.

Result    Returns 0 upon success or an error if unsuccessful.

Comments   This function is called in two cases: when a text field item is being drawn (TblDrawTable or TblRedrawTable) and when a text field item is being selected (part of TblHandleEvent’s handling of tblEnterEvent). If this function returns an error (any nonzero value) and the item is being selected, then the item is not selected and the table’s editing attribute is set to false.

You return the same values for dataH, dataOffset, and dataSize that you would pass to FldSetText. That is, you can use this function to point the table cell’s text field to a string in a database record so that you can edit that string directly using text field routines. To do so, return the handle to a database record in dataH, the offset from the start of the record to the start of the string in dataOffset, and the allocated size of the string in dataSize.

The handle that you return from this function is assumed to contain a null-terminated string starting at dataOffset bytes in the memory chunk. The string should be between 0 and dataSize - 1 bytes in length.

As with FldSetText, you are responsible for freeing the memory associated with the dataH parameter. You can do so in the TableSaveDataFuncType function, but it is only called for a cell that has been edited. For non-editable text cells or text cells that are editable but were never selected, free the memory when you close the form.

The fld pointer passed to your function has already been initialized with default values by the table code. If you want to override a field’s attributes (for example, if you want to change the underline mode) you can do so in this function.

See Also   TblDrawTable, TblHandleEvent, TableLoadDataFuncType
TableSaveDataFuncType

**Purpose**  
Save the data associated with a text field.

**Declared In**  
Table.h

**Prototype**  
Boolean TableSaveDataFuncType (void *tableP, Int16 row, Int16 column)

**Parameters**  
- `tableP`  
  Pointer to a table object. (See `TableType`.)
- `row`  
  Row number of the table item to load.
- `column`  
  Column number of the table item to load.

**Result**  
Return `true` if the table should be redrawn, or `false` if the table does not need to be redrawn.

**Comments**  
This is called before the memory associated with the currently selected text field in a table cell is freed. Implement this function if you need to do any special cleanup before this memory is freed.

This function is called only when the currently selected editable text field is releasing the focus. You can use `TblGetCurrentField` to retrieve a pointer to this field. It is called only on the currently selected field, not on any other fields in the table.

Note that the table manager already disassociates the memory handle from the text field for you so that the memory associated with your data is not freed when the field is freed. The table manager also calls `FldCompactText` for you.

If the text handle you returned in your `TableLoadDataFuncType` callback points to a string on the dynamic heap, you should implement this callback function to store or free the handle. You can use `FldGetTextHandle` to obtain the handle.

If you return `true` from this function, `TblRedrawTable` is called. You should mark invalid any table rows that you want redrawn before returning.

**See Also**  
`TblSetSaveDataProcedure`
UI Color List

This chapter provides information about the UI color list by discussing the following topics:

- UI Color Data Types
- UI Color Functions

The header file UIColor.h declares the API that this chapter describes. For more information on the color list, see “Color and Grayscale Support” on page 144 in the Palm OS Programmer’s Companion, vol. I.

UI Color Data Types

UIColorTableEntries

The UIColorTableEntries enum declares symbolic color constants for the various UI elements.

Do not confuse the UI color list with the system color table. The system color table (or system palette) defines all available colors for the display or draw window, whether they are in use or not. The UI color list defines the colors used to draw the interface objects.

typedef enum UIColorTableEntries {
    UIOBJECTFRAME = 0,
    UIOBJECTFILL,
    UIOBJECTFOREGROUND,
    UIOBJECTSELECTEDFILL,
    UIOBJECTSELECTEDFOREGROUND,
    
    UIMENUFFrame,
    UIMENUFFill,
    UIMENUFOREGROUND,
    UIMENUSELECTEDFILL,
    UIMENUSELECTEDFOREGROUND,
}
UI Color Data Types

- UIFieldBackground
- UIFieldText
- UIFieldTextLines
- UIFieldCaret
- UIFieldTextHighlightBackground
- UIFieldTextHighlightForeground
- UIFieldFepRawText
- UIFieldFepRawBackground
- UIFieldFepConvertedText
- UIFieldFepConvertedBackground
- UIFieldFepUnderline

- UIFormFrame
- UIFormFill

- UIDialogFrame
- UIDialogFill

- UIAlertFrame
- UIAlertFill

- UIOK
- UICaution
- UICWarning

- UILastColorTableEntry

} UIColorTableEntries;
### Field Descriptions

<table>
<thead>
<tr>
<th>Field Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UIObjectFrame</td>
<td>Color for the border of user interface objects (such as command buttons, push buttons, selector triggers, menus, arrows checkboxes, and other controls).</td>
</tr>
<tr>
<td>UIObjectFill</td>
<td>The background color for a solid or “filled” user interface object.</td>
</tr>
<tr>
<td></td>
<td>Note that UI objects in tables use the UIField... colors instead of the UIObject... colors.</td>
</tr>
<tr>
<td>UIObjectForeground</td>
<td>The color for foreground items (such as labels or graphics) in a user interface object.</td>
</tr>
<tr>
<td>UIObjectSelectedFill</td>
<td>The background color of the currently selected user interface object, whether that object is solid or not.</td>
</tr>
<tr>
<td>UIObjectSelectedForeground</td>
<td>The color of foreground items in a selected user interface object.</td>
</tr>
<tr>
<td>UIMenuFrame</td>
<td>The color of the border around the menu.</td>
</tr>
<tr>
<td>UIMenuFill</td>
<td>The background color of a menu item.</td>
</tr>
<tr>
<td>UIMenuForeground</td>
<td>The color of the menu’s text.</td>
</tr>
<tr>
<td>UIMenuSelectedFill</td>
<td>The background color of a selected menu item.</td>
</tr>
<tr>
<td>UIMenuSelectedForeground</td>
<td>The color of the text of a selected menu item.</td>
</tr>
<tr>
<td>UIFieldBackground</td>
<td>The background color of an editable text field.</td>
</tr>
<tr>
<td>UIFieldText</td>
<td>The color of the text in the editable field.</td>
</tr>
<tr>
<td>UIFieldTextLines</td>
<td>The color of underlines in an editable field.</td>
</tr>
<tr>
<td>UIFieldCaret</td>
<td>The color of the cursor in an editable text field.</td>
</tr>
<tr>
<td>UIFieldTextHighlightBackground</td>
<td>The background color for selected text in an editable text field.</td>
</tr>
</tbody>
</table>
**UI Color List**

*UI Color Data Types*

<table>
<thead>
<tr>
<th>UI Color Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UIFieldTextHighlightForeground</td>
<td>The color of the selected text in an editable text field.</td>
</tr>
<tr>
<td>UIFieldFepRawText</td>
<td>Color used for unconverted text in the inline conversion area when a FEP is used as a text input method (for example, on Japanese devices).</td>
</tr>
<tr>
<td></td>
<td>If the FEP colors are identical to field colors, unconverted text has a solid underline.</td>
</tr>
<tr>
<td>UIFieldFepRawBackground</td>
<td>The background color for unconverted text in the inline conversion area when a FEP is used as a text input method.</td>
</tr>
<tr>
<td></td>
<td>If the FEP colors are identical to field colors, unconverted text has a solid underline.</td>
</tr>
<tr>
<td>UIFieldFepConvertedText</td>
<td>Color used for converted text in the inline conversion area when a FEP is used as a text input method (for example, on Japanese devices).</td>
</tr>
<tr>
<td></td>
<td>If the FEP colors are identical to field colors, converted text has a double-thick underline.</td>
</tr>
<tr>
<td>UIFieldFepConvertedBackground</td>
<td>The background color used for converted text in the inline conversion area.</td>
</tr>
<tr>
<td></td>
<td>If the FEP colors are identical to field colors, converted text has a double-thick underline.</td>
</tr>
<tr>
<td>UIFieldFepUnderline</td>
<td>The color used for underlines in the inline conversion area.</td>
</tr>
<tr>
<td>UIFormFrame</td>
<td>The color of the border and titlebar on a form.</td>
</tr>
<tr>
<td>UIFormFill</td>
<td>The background color of a form. White is recommended for this value.</td>
</tr>
<tr>
<td>UIDialogFrame</td>
<td>The color of a border and titlebar on a modal form.</td>
</tr>
</tbody>
</table>
UI Color Functions

UIColorGetTableEntryIndex

**Purpose**
Return the index value for a UI color for the current system palette.

**Declared In**
UIColor.h

**Prototype**
IndexedColorType UIColorGetTableEntryIndex
(UIColorTableEntries which)

**Parameters**
- `which` One of the symbolic color constants. See UIColorTableEntries.

**Result**
Returns the system color table index of the color used for the specified symbolic color.

**Comments**
One way to find out the indexes of all the colors that the OS is using is to loop through the UI color list, calling

**Palm OS®  does not currently use the UIOK, UICaution, and UIWarning constants.**

**Compatibility**
Implemented only if 3.5 New Feature Set is present.
UIColorGetTableEntryIndex for each slot, and keep a list (excluding duplicates).
IndexedColorType
colorsUsed[UILastColorTableEntry];
UInt16 numColors = 0;
...
for (i = 0; i < UILastColorTableEntry; i++) {
    IndexedColorType currentColor;
    Boolean isNew = true;

    currentColor = UIColorGetTableEntryIndex(i);

    for (j = 0; ((j < numColors) && isNew); j++)
        if (colorsUsed[j] == currentColor)
            isNew = false; /* exit loop */
    if (isNew) {
        numColors++;
        colorsUsed[j] = currentColor;
    }
}

To get the RGB values of the colors, do the same thing but call UIColorGetTableEntryRGB.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also IndexedColorType, WinIndexToRGB

UIColorGetTableEntryRGB

Purpose Return the RGB value for the UI color.

Declared In UIColor.h

Prototype void UIColorGetTableEntryRGB
(UIColorTableEntries which, RGBColorType *rgbP)

Parameters -> which One of the symbolic color constants. See UIColorTableEntries.
<- rgbP Pointer to an RGB color value corresponding to the current color used for the symbolic color. (See RGBColorType.)
UI Color List
UI Color Functions
UIColorSetTableEntry

Purpose
Change a value in the UI color list.

Declared In
UIColor.h

Prototype
Err UIColorSetTableEntry
(UIColorTableEntries which,
const RGBColorType *rgbP)

Parameters
-> which One of the symbolic color constants. See UIColorTableEntries.
-> rgbP The RGB value of the color that should be used for the specified UI object. (See RGBColorType.)

Result
Returns 0 upon success.

Comments
Sets the value of a UI color entry to the passed RGB value. Updates the index for that UI color entry to the current best fit for that RGB value according to the palette used by the current draw window.

It is best to use this function only if the draw window is currently onscreen. Otherwise, the best-fit algorithm may choose a color that is not available on the current screen.

See Also
WinIndexToRGB, UIColorGetTableEntryIndex, UIColorGetTableEntryRGB

Result
Returns nothing.

Comments
In general, it is more efficient to work with indexed color entries instead of RGB color entries.

Compatibility
Implemented only if 3.5 New Feature Set is present.

See Also
UIColorGetTableEntryIndex, WinRGBToIndex
UI Color List
UI Color Functions
UI Controls

This chapter describes the UI controls API as declared in UIControls.h.

UI Control Functions

UIBrightnessAdjust

Purpose  Displays the brightness adjust dialog.

Declared In  UIControls.h

Prototype  void UIBrightnessAdjust()

Parameters  None

Result  Returns nothing.

Comments  On hardware that supports a brightness setting, this function displays a dialog that allows the user to change the brightness level. On hardware that has a backlight, this function toggles the backlight.

Compatibility  Implemented only if 3.5 New Feature Set is present.
**UIContrastAdjust**

**Purpose**
Displays the contrast adjust dialog (currently only available on the Palm V™ Connected Organizer).

**Declared In**
UIControls.h

**Prototype**
void UIContrastAdjust()

**Parameters**
None.

**Result**
Returns nothing.

**Compatibility**
This function was renamed from ContrastAdjust to UIContrastAdjust in Palm OS® release 3.5. The ContrastAdjust function is available if 3.1 New Feature Set is present.

**UIPickColor**

**Purpose**
Displays a dialog that allows the user to choose a color.

**Declared In**
UIControls.h

**Prototype**
Boolean UIPickColor (IndexedColorType *indexP, RGBColorType *rgbP, UIPickColorStartType start, const Char *titleP, const Char *tipP)

**Parameters**
<-> indexP

Index value of the selected color. (See IndexedColorType.) Upon entry, this points to the index value of the color initially selected. Upon return, this points to the index value of the color the user selected. Pass NULL to not set or return this value.
UI Controls

UI Control Functions

- **rgbP**: RGB value of the selected color. (See `RGBColorType`.) Upon entry, this points to the RGB value of the color initially selected when the dialog is displayed. Upon return, this points to the RGB value that the user selected. Pass `NULL` to not set or return this value.

- **start**: Either `UIPickColorStartPalette` to display the system palette as a series of color squares or `UIPickColorStartRGB` to display individual sliders for the red, green, and blue values. This parameter is only used if both `indexP` and `rgbP` are not `NULL`.

- **titleP**: String to display as the title of the dialog. Specify `NULL` to use the default title, which is “Pick Color.”

- **tipP**: Not used.

**Result**

Returns `true` if a new color was selected, `false` otherwise.

**Comments**

Use this function to allow users to choose a color used in your user interface. (The system never calls `UIPickColor`.)

This function can display two versions of the dialog: palette or RGB. The palette version of the dialog displays a series of squares, each containing a different color defined on the system palette. The `indexP` value contains the index of the square that is initially selected.

The RGB version of the dialog displays three sliders that allow the user to select the level of red, green, and blue in the color. The `rgbP` parameter contains the red, green, and blue values initially shown in the dialog. The sliders only allow values that are defined in the current system color table.

If `indexP` is initially `NULL`, only the RGB dialog is displayed. Similarly, if `rgbP` is `NULL`, only the palette version is displayed. If both parameters are non-`NULL`, the system adds a pull-down list that allows the user to switch between the palette dialog and the RGB dialog, and the `start` parameter controls which version of the
dialog is initially shown. In this case, both indexP and rgbP contain the value of the user-selected color upon return.

Palm OS 3.5 supports a maximum of 256 colors. The number of possible RGB colors greatly exceeds this amount. For this reason, the chosen RGB may not have an exact match. If this is the case, the indexP parameter (if not NULL) contains the closest match using a luminance best-fit if the color lookup table is entirely grayscale (red, green, and blue values for each entry are identical), or a shortest-distance fit in the RGB space is used if the palette contains colors.

**Compatibility**

Implemented only if [3.5 New Feature Set](#) is present.

**See Also**

WinSetBackColor, WinSetForeColor, WinSetTextColor, UIColorColorSetTableEntry
Miscellaneous User Interface Functions

This chapter provides descriptions of miscellaneous user interface functions. It covers the following topics:

- **Miscellaneous User Interface Data Structures**
- **Miscellaneous User Interface Functions**

You can find declarations for the functions described in this chapter in the header files `AppLaunchCmd.h`, `PhoneLookup.h`, and `UIResources.h`.

### Miscellaneous User Interface Data Structures

The `PhoneNumberLookupCustom` function uses these data structures to look up contact information based upon the current cursor position.

#### AddressLookupFields

The `AddressLookupFields` enum specifies the fields you can search by and the corresponding fields to return using the `field1` and `field2` elements of the `AddrLookupParamsType` structure. For both `field1` and `field2` pass one of the values up to, but not including, `addrLookupFieldCount`.

```c
typedef enum {
    addrLookupName,  // Name
    addrLookupFirstName,  // First name
    addrLookupCompany,  // Company
    addrLookupAddress,  // Address
    addrLookupCity,     // City
    addrLookupState,    // State
    addrLookupZipCode,  // Zip Code
    // More fields...
} AddressLookupFields;
```
addrLookupCountry,
addrLookupTitle,
addrLookupCustom1,
addrLookupCustom2,
addrLookupCustom3,
addrLookupCustom4,
addrLookupNote,
addrLookupWork,
addrLookupHome,
addrLookupFax,
addrLookupOther,
addrLookupEmail,
addrLookupMain,
addrLookupPager,
addrLookupMobile,
addrLookupSortField,
addrLookupListPhone,
addrLookupFieldCount,

    addrLookupNoField = 0xff
} AddressLookupFields;

**AddrLookupParamsType**

Pass this structure to PhoneNumberLookupCustom to precisely control the phone number lookup dialog and paste process.

```c
typedef struct {
    Char *title;
    Char *pasteButtonText;
    Char lookupString[addrLookupStringLength];
    AddressLookupFields field1;
    AddressLookupFields field2;
    Boolean field2Optional;
    Boolean userShouldInteract;
    Char *formatStringP;
    MemHandle resultStringH;
    UInt32 uniqueID;
} AddrLookupParamsType;
```
typedef AddrLookupParamsType
*AddrLookupParamsPtr;

**Value Descriptions**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>Title to appear in the title bar. Supply NULL to use the default title.</td>
</tr>
<tr>
<td>pasteButtonText</td>
<td>Text to appear in paste button. Supply NULL to use the default, “paste”.</td>
</tr>
<tr>
<td>lookupString</td>
<td>Buffer containing the string to look up. If the string matches only one record, that record is used without presenting the lookup dialog to the user. PhoneNumberLookup and PhoneNumberLookupCustom both set this field based upon the current selection or cursor position.</td>
</tr>
<tr>
<td>field1</td>
<td>Field to search by. This field appears on the left side of the lookup dialog. If the field is the sort field, the search is performed using a binary search. If the field isn’t the sort field, searching is performed by a linear search, which can be slow. Supply one of the values in the AddressLookupFields enum to specify the field to search by.</td>
</tr>
<tr>
<td>field2</td>
<td>Field to display on the right. Often displays some information about the person. If it is a phone field and a record has multiple instances of the phone type then the person appears once per instance of the phone type. Either field1 or field2 may be a phone field, but not both. Supply one of the values in the AddressLookupFields enum to specify the field to display.</td>
</tr>
</tbody>
</table>
Miscellaneous User Interface Functions

Miscellaneous User Interface Data Structures

field2Optional A value of true means that the record need not have field2 in order to be listed. A value of false indicates that field2 is required in order for the record to be listed.

userShouldInteract A value of true forces the user to resolve non-unique lookups. A false value means a non-unique and complete lookup causes resultStringH to be set to NULL and uniqueID to be set to 0.
formatStringP  Controls the format of the paste string. All characters in the format string are literal unless they identify a field (signified by a caret (^) followed by the field name). For example, the format string “^first - ^home” might result in “Roger - 123-4567”. Allowable field names are:

- name
- first
- company
- address
- city
- state
- zipcode
- country
- title
- custom1
- custom2
- custom3
- custom4
- work
- home
- fax
- other
- email
- main
- pager
- mobile
- listname
PhoneNumberLookup

Purpose
Calls the Address Book application to look up a phone number.

Declared In
PhoneLookup.h

Prototype
void PhoneNumberLookup (FieldType *fldP)

Parameters
-> fldP Field object in which the text to match is found.

Result
Nothing returned; it’s locked.

Comments
This function displays the user’s phone list and inserts the chosen name and number (or company name, name, and number, if that’s how the user’s Address Book preferences indicate that the phone list should be sorted) into the specified field. When displaying the phone list, PhoneNumberLookup scrolls the list to that entry that best matches the supplied field. The match compares the field contents against the name or company name (depending on the user’s preferences) as follows:

- If the field contains selected text, PhoneNumberLookup tries to match against the selected text. The selected text is then replaced with the text of the chosen address list entry.
- If there is no selected text in the field, PhoneNumberLookup matches against the word in which the cursor lies (the match will take place if the cursor is at the beginning, the end, or within a word). The matched word is replaced with the text of the chosen address list entry.
• If the cursor does not lie within or adjoin a word, PhoneNumberLookup displays the address list starting at the first entry, and the text of the chosen entry is inserted at the current position within the text field.

If the user chooses Cancel when the address list is displayed, the field contents are left unaltered. The paste operation takes place through the clipboard so that Undo can be used to restore the field to its previous state.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also PhoneNumberLookupCustom

PhoneNumberLookupCustom

Purpose Calls the Address Book application to look up a phone number.

Declared In PhoneLookup.h

Prototype void PhoneNumberLookupCustom (FieldType *fldP,
AddrLookupParamsType *params,
Boolean useClipboard)

Parameters
- > fldP Field object in which the text to match is found.
<-> params A structure that allows full control over the find dialog and the format of the resulting paste string. See AddrLookupParamsType for a description of the fields in this structure.

- > useClipboard If true, PhoneNumberLookupCustom pastes the result into the field through the clipboard, thereby enabling undo.

Result Nothing returned; it’s locked.
Comments

This function displays two fields from each record in the user’s address list and inserts a formatted string based upon fields in the chosen record into the specified field. When displaying the address list, PhoneNumberLookupCustom scrolls the list to that entry that best matches the supplied field. The match compares the field contents against the field specified in the params structure’s field1 element as follows:

- If the field contains selected text, PhoneNumberLookup tries to match against the selected text. The selected text is then replaced with the text of the chosen address list entry.
- If there is no selected text in the field, PhoneNumberLookup matches against the word in which the cursor lies (the match will take place if the cursor is at the beginning, the end, or within a word). The matched word is replaced with the text of the chosen address list entry.
- If the cursor does not lie within or adjoin a word, PhoneNumberLookup displays the address list starting at the first entry, and the text of the chosen entry is inserted at the current position within the text field.

PhoneNumberLookupCustom copies the portion of the field used for the search—the selected text or the word in which the cursor lies—into the lookupString field in the params structure prior to replacing that part of the field with the user-selected entry.

If the user chooses Cancel when the address list is displayed, the field contents are left unaltered. Depending on the value of the useclipboard parameter, the paste operation can take place through the clipboard so that Undo can be used to restore the field to its previous state.

Compatibility

Implemented only if 4.0 New Feature Set is present.
ResLoadConstant

**Purpose**
Load a constant from a 'tint' resource and return its value.

**Declared In**
UIResources.h

**Prototype**
UInt32 ResLoadConstant (UInt16 rscID)

**Parameters**
rscID
The ID of the 'tint' resource (symbolically named constantRscType) to load.

**Result**
The four-byte value of the constant in the resource, or 0 if the resource could not be found. The return value may be cast as necessary.

**Comments**
Use this function to load constant values that are stored as 'tint' resources. (All open resource databases are searched for the resource ID you specify.) You should store a constant value as a resource when its value changes depending on the locale.

As an example, consider the maximum length of the Alarm Sound trigger label in the Datebook application’s preferences panel. The list displayed by this trigger uses the localized name for each sound stored in the system. Because localized names are used, the maximum length that the Datebook application allows for the label differs depending on the current locale. The maximum length is stored as a resource constant so that each overlay database can specify a different value for the constant.

**Compatibility**
Implemented only if 3.5 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS®, link with the PalmOSGlue library and call ResGlueLoadConstant. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**
DmGetResource, DmGet1Resource
ResLoadForm

Purpose   Copy and initialize a form resource. The structures are complete except pointers updating. Pointers are stored as offsets from the beginning of the form.

Declared In  UIResources.h

Prototype  void* ResLoadForm (UInt16 rscID)

Parameters  - > rscID       The resource ID of the form.

Result   The handle of the memory block that the form is in, since the form structure begins with the WindowType, this is also a WinHandle.

Compatibility  If 5.0 New Feature Set is present this function is unimplemented.

ResLoadMenu

Purpose   Copy and initialize a menu resource. The structures are complete except pointers updating. Pointers are stored as offsets from the beginning of the menu.

Declared In  UIResources.h

Prototype  void* ResLoadMenu (UInt16 rscID)

Parameters  - > rscID       The resource ID of the menu.

Result   The handle of the memory block that the form is in, since the form structure begins with the WindowType this is also a WinHandle.
Part II: System Management
Alarm Manager

This chapter provides reference material for the alarm manager:

- Alarm Manager Functions
- Application-Defined Functions

The alarm manager API is declared in the header file AlarmMgr.h.

For more information on the Alarm Manager, see the section “Alarms” in the Palm OS Programmer’s Companion, vol. I.

Alarm Manager Functions

**AlmGetAlarm**

**Purpose**
Return the date and time for the application’s alarm, if one is set.

**Declared In**
AlarmMgr.h

**Prototype**
UInt32 AlmGetAlarm (UInt16 cardNo, LocalID dbID, UInt32* refP)

**Parameters**
- cardNo Number of the storage card on which the application resides.
- dbID Local ID of the application.
<refP>
The alarm’s reference value is returned here. This value is passed to the application when the alarm is triggered.

**Result**
The date and time the alarm will trigger, given in seconds since 1/1/1904; if no alarm is active for the application, 0 is returned for the alarm seconds and the reference value is undefined.

**See Also**  
[AlmSetAlarm](#)

---

**AlmGetProcAlarm**

**Purpose**
Macro that returns the date and time that a procedure alarm will trigger. Also returns the caller-defined alarm reference value.

**Declared In**
AlarmMgr.h

**Prototype**
AlmGetProcAlarm (procP, refP)

**Parameters**
- procP: Pointer to a function that will be called when alarm is triggered. See [AlmAlarmProcPtr](#).

- refP: A UInt32 pointer to a location where the alarm’s reference value is returned. This value is passed to the procedure when the alarm is triggered.

**Result**
The date and time the alarm will trigger, given in seconds since 1/1/1904; if no alarm is active for the procedure, 0 is returned for the alarm seconds and the reference value is undefined.

**Compatibility**
Implemented only if 3.2 New Feature Set is present.

**See Also**  
[AlmSetProcAlarm](#)
**AlmSetAlarm**

**Purpose**  Set or cancel an alarm for the given application.

**Declared In**  AlarmMgr.h

**Prototype**  `Err AlmSetAlarm (UInt16 cardNo, LocalID dbID, UInt32 ref, UInt32 alarmSeconds, Boolean quiet)`

**Parameters**

- `-> cardNo`  Number of the storage card on which the application resides.
- `-> dbID`  Local ID of the application.
- `-> ref`  Caller-defined value. This value is passed with the launch code that notifies the application that the alarm has been triggered.
- `-> alarmSeconds`  Alarm date/time in seconds since 1/1/1904, or 0 to cancel the current alarm (if any).
- `-> quiet`  Reserved for future upgrade. This value is not currently used.

**Result**

- `0`  No error.
- `almErrMemory`  Insufficient memory.
- `almErrFull`  Alarm table is full.

**Comments**  This function sets an alarm for the specified application. An application can have only one alarm set at a time. If an alarm for this application has already been set, it is replaced with the new alarm. The `alarmSeconds` parameter specifies the time at which the alarm will be triggered. As soon as possible after this time, the alarm manager sends the `sysAppLaunchCmdAlarmTriggered` launch code to the specified application. If there is another alarm that should be set for this application, you can set it in response to this launch code. Following the `sysAppLaunchCmdAlarmTriggered` launch code, the alarm manager sends a `sysAppLaunchCmdDisplayAlarm` launch code. This is where your application should do things such as display a modal dialog.
indicating that the event has occurred. Read more about these launch codes in Chapter 1, “Application Launch Codes.”

If your application needs access to any particular value to respond to the alarm, pass a pointer to that value in the ref parameter. The system will pass this pointer back to the application in the launch codes’ parameter blocks.

See Also  
AlmGetAlarm

**AlmSetProcAlarm**

**Purpose**  
Macro that sets or cancels a procedure alarm.

**Declared In**  
AlarmMgr.h

**Prototype**  
AlmSetProcAlarm (procP, ref, alarmSeconds)

**Parameters**
- -> procP  
  Pointer to a function that should be called when alarm is triggered. See AlmAlarmProcPtr.
- -> ref  
  A caller-defined UInt32 value. This value is passed with the launch code that notifies the application that the alarm has been triggered.
- -> alarmSeconds  
  A UInt32 indicating the alarm date/time in seconds since 1/1/1904, or 0 to cancel the current alarm (if any).

**Result**  
Returns one of the following error codes:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error.</td>
</tr>
<tr>
<td>almErrMemory</td>
<td>Insufficient memory.</td>
</tr>
<tr>
<td>almErrFull</td>
<td>Alarm table is full.</td>
</tr>
</tbody>
</table>

**Comments**  
This macro is similar to the AlmSetAlarm function, but it specifies a procedure to be called at the specified date and time rather than an application to be launched. With this macro, you can set alarms that are independent of any application. For example, a shared library
can set procedure alarms that call a procedure implemented in the library.

Procedure alarms also differ from regular system alarms in that if they trigger when the device is in sleep mode, the LCD does not light up. Thus, you can use procedure alarms to perform a scheduled task in a manner that is entirely hidden from the user.

**IMPORTANT:** Because the procP pointer is used to directly call the procedure, the pointer must remain valid from the time AlmSetProcAlarm is called to the time the alarm is triggered. If the procedure is in a shared library, you must keep the library open. If the procedure is in a separately loaded code resource, the resource must remain locked until the alarm is triggered. When you close a library or unlock a resource, you must remove any pending alarms. If you don’t, the system will crash when the alarm is triggered.

**Compatibility**
Implemented only if 3.2 New Feature Set is present.

**See Also**
AlmGetProcAlarm

### Application-Defined Functions

#### AlmAlarmProcPtr

**Purpose**
A procedure to be executed when an alarm is triggered.

**Declared In**
AlarmMgr.h

**Prototype**
void (*AlmAlarmProcPtr) (UInt16 almProcCmd, SysAlarmTriggeredParamType *paramP)

**Parameters**
- `-> almProcCmd`: One of the AlmProcCmdEnum constants. These are commands that your function must handle. Possible values are:
alarmProcCmdTriggered

The alarm’s date and time has passed and the alarm has been triggered. The function should perform its main task in response to this command.

alarmProcCmdReschedule

A system time change occurred, so the function must reschedule the alarm.

- paramP Pointer to a SysAlarmTriggeredParamType structure. See below.

**Result**

Returns nothing.

**Comments**

`AlmAlarmProcPtr` procedures are called when an alarm set by `AlmSetProcAlarm` is triggered. Your implementation should check the value of `alarmProcCmd` and respond accordingly.

The `paramP` parameter is a pointer to a `SysAlarmTriggeredParamType` structure. This structure is defined as:

```c
typedef struct SysAlarmTriggeredParamType {
    UInt32   ref;
    UInt32   alarmSeconds;
    Boolean  purgeAlarm;
} SysAlarmTriggeredParamType;
```

The `ref` and `alarmSeconds` are both values specified in `AlmSetProcAlarm` when the alarm is set. The `purgeAlarm` field specifies if the alarm will be removed from the alarm table when the function returns so that the `sysAppLaunchCmdDisplayAlarm` launch code is not triggered. This should be `true` for all procedure alarms; the alarm manager set it to `true` for you after your function returns.

If necessary, you can define new values for the `alarmProcCmd` parameter to call the procedure for something other than a triggered alarm or a system time change. The value you use must be greater than the constant `alarmProcCmdCustom` as defined in `AlarmMgr.h`.
**Compatibility**  Implemented only if [3.2 New Feature Set](#) is present.

**See Also**  [AlmGetProcAlarm](#)
Bitmaps

This chapter provides information about bitmaps by discussing these topics:

- Bitmap Data Structures
- Bitmap Constants
- Bitmap Resources
- Bitmap Functions

The header file `Bitmap.h` declares the API that this chapter describes. For more information on bitmaps, see the section “Bitmaps” on page 123 in the Palm OS Programmer’s Companion, vol. I.

**Bitmap Data Structures**

**BitmapCompressionType**

The `BitmapCompressionType` enum specifies possible bitmap compression types. These are the possible values for the `compressionType` field of `BitmapType`. You can compress or uncompress a bitmap using a call to `BmpCompress`.

```c
typedef enum {
    BitmapCompressionTypeScanLine = 0,
    BitmapCompressionTypeRLE,
    BitmapCompressionTypePackBits,
    BitmapCompressionTypeEnd,
    BitmapCompressionTypeBest = 0x64,
    BitmapCompressionTypeNone = 0xFF
} BitmapCompressionType;
```
## Bitmaps

### Bitmap Data Structures

#### Value Descriptions

<table>
<thead>
<tr>
<th>BitmapCompressionTypeScanLine</th>
<th>Use scan line compression. Scan line compression is compatible with Palm OS® 2.0 and higher.</th>
</tr>
</thead>
<tbody>
<tr>
<td>BitmapCompressionTypeRLE</td>
<td>Use RLE compression. RLE compression is supported in Palm OS 3.5 and higher.</td>
</tr>
<tr>
<td>BitmapCompressionTypePackBits</td>
<td>Use PackBits compression. PackBits compression is supported in Palm OS 4.0 only.</td>
</tr>
<tr>
<td>BitmapCompressionTypeEnd</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>BitmapCompressionTypeBest</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>BitmapCompressionTypeNone</td>
<td>No compression is used.</td>
</tr>
</tbody>
</table>

This value should only be used as an argument to BmpCompress.

#### Compatibility

BitmapCompressionType is only defined if 3.5 New Feature Set is present. Earlier releases do support compressed bitmaps, but in scan line format only.

### BitmapDirectInfoType

For direct color bitmaps—each pixel is represented by an RGB triplet rather than a palette index—the BitmapDirectInfoType structure follows the color table if one is present, or immediately follows the BitmapType if a color table is not present. For direct color bitmaps, only 16 bits per pixel is supported, 5 bits for red, 6 bits for green, and 5 bits for blue.
WARNING! This structure is documented so that you can directly access the internals of your own bitmap resources. *Bitmaps created by Palm OS are not guaranteed to adhere to this structure*; you cannot cast a direct color bitmap data pointer from a bitmap created by the Palm OS to this structure and expect to be able to correctly access the structure’s fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

```c
typedef struct BitmapDirectInfoType {
    UInt8 redBits;
    UInt8 greenBits;
    UInt8 blueBits;
    UInt8 reserved;
    RGBColorType transparentColor;
} BitmapDirectInfoType;
```

**Field Descriptions**

- **redBits**
  Number of bits used by the red component in each pixel.

- **greenBits**
  Number of bits used by the green component in each pixel.

- **blueBits**
  Number of bits used by the blue component in each pixel.

- **reserved**
  Must be zero. Reserved for future use.

- **transparentColor**
  Contains the red, green, and blue components of the transparent color.

**Compatibility**

*BitmapDirectInfoType* is only defined if 4.0 New Feature Set is present.

**BitmapFlagsType**

The *BitmapFlagsType* bit field defines the flags field of *BitmapType*. It specifies the bitmap’s attributes.
**WARNING!** This structure is documented so that you can directly access the internals of your own bitmap resources. *Bitmaps created by Palm OS are not guaranteed to adhere to this structure*; you cannot cast the `flags` field of a bitmap created by the Palm OS to this structure and expect to be able to correctly access the structure’s fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

```c
typedef struct BitmapFlagsType {
    UInt16 compressed:1;
    UInt16 hasColorTable:1;
    UInt16 hasTransparency:1;
    UInt16 indirect:1;
    UInt16 forScreen:1;
    UInt16 directColor:1;
    UInt16 indirectColorTable:1;
    UInt16 noDither:1;
    UInt16 reserved:8;
} BitmapFlagsType;
```

**Field Descriptions**

**compressed**  If true, the bitmap is compressed and the `compressionType` field specifies the compression used. If false, the bitmap is uncompressed. The `BmpCompress` function sets this field.

**hasColorTable**  If true, the bitmap has its own color table. If false, the bitmap uses the system color table. You specify whether the bitmap has its own color table when you create the bitmap.
hasTransparency  If true, the OS will not draw pixels that have a value equal to the transparentIndex. If false, the bitmap has no transparency value. You specify the transparent color when you create the bitmap using Constructor, or you can specify it programmatically with BmpSetTransparentValue. To obtain the value of this field, call BmpGetTransparentValue.

indirect  If true, the address to the bitmap’s data is stored where the bitmap itself would normally be stored. The actual bitmap data is stored elsewhere. If false, the bitmap data is stored directly following the bitmap header or directly following the bitmap’s color table if it has one. Never set this flag.

Note that this flag is supported for bitmaps created by Palm OS only; this flag is not used in user-created bitmap resources.

forScreen  If true, bitmap intended for the display (screen) window. Never set this flag.

Note that this flag is supported for bitmaps created by Palm OS only; this flag is not used in user-created bitmap resources.

directColor  If true, bitmap is a direct color (RGB) bitmap.
Bitmaps

Bitmap Data Structures

**indirectColorTab**
If true, and if hasColorTable is true, a pointer to the bitmap’s color table immediately follows the `BitmapType` structure. If false, and hasColorTable is true, the color table immediately follows the `BitmapType` structure. If hasColorTable is false, `indirectColorTable` is ignored. The indirect bit uses similar logic: if both the color table and the bitmap data are indirect, the color table pointer precedes the bitmap data pointer.

Note that this flag is supported for bitmaps created by Palm OS only; this flag is not used in user-created bitmap resources.

**noDither**
If true, the blitter does not dither the bitmap. If false, the source bitmap is dithered if it has a bit depth greater than the destination bitmap.

**reserved**
Reserved for future use.

**Compatibility**
The hasTransparency, indirect, and forScreen flags are only defined if 3.5 New Feature Set is present. The directColor flag is only defined if 4.0 New Feature Set is present. The indirectColorTable and noDither flags are only defined if the High-Density Display Feature Set is present.

**BitmapPtr**
The `BitmapPtr` type defines a pointer to a `BitmapType` structure.

```c
typedef BitmapType *BitmapPtr;
```

**BitmapType**
The `BitmapType` structure represents that which is common to all `BitmapTypeVx` structures (`BitmapTypeV0`, `BitmapTypeV1`, `BitmapTypeV2`, and `BitmapTypeV3`). The `BitmapType` structures define both the bitmaps representing the window display and
bitmap resources ('Tbmp' and 'tAIB') that you create using Constructor or some other application and load into your program. Because BitmapType is merely a portion of the BitmapTypeVx structures, you should never do sizeof(BitmapType).

**WARNING!** This structure is documented so that you can directly access the internals of your own bitmap resources. *Bitmaps created by Palm OS are not guaranteed to adhere to this structure*; you cannot cast a bitmap created by the Palm OS to this structure and expect to be able to correctly access the structure’s fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

```c
typedef struct BitmapType {
    Int16 width;
    Int16 height;
    UInt16 rowBytes;
    BitmapFlagsType flags;
    UInt8 pixelSize;
    UInt8 version;
} BitmapType;

typedef BitmapType* BitmapPtr;
```

**Field Descriptions**

- **width**  
  The width of the bitmap in pixels. You specify this value when you create the bitmap. Use `BmpGetDimensions` to access this field.

- **height**  
  The height of the bitmap in pixels. You specify this value when you create the bitmap. Use `BmpGetDimensions` to access this field.

- **rowBytes**  
  The number of bytes stored for each row of the bitmap where height is the number of rows. Use `BmpGetDimensions` to obtain the contents of this field.
### Bitmaps

#### Bitmap Data Structures

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>The bitmap’s attributes. See <code>BitmapFlagsType</code>.</td>
</tr>
<tr>
<td>pixelSize</td>
<td>The pixel depth. Currently supported pixel depths are 1, 2, 4, and 8-bit. You specify this value when you create the bitmap. Use <code>BmpGetBitDepth</code> to access the contents of this field.</td>
</tr>
<tr>
<td>version</td>
<td>The version of bitmap encoding used. See “Bitmap Constants” on page 535. The value in this field determines the data structure to use when interpreting the fields following version: a value of <code>BitmapVersionZero</code> (0) corresponds to <code>BitmapTypeV0</code>, <code>BitmapVersionOne</code> (1) corresponds to <code>BitmapTypeV1</code>, <code>BitmapVersionTwo</code> (2) corresponds to <code>BitmapTypeV2</code>, and <code>BitmapVersionThree</code> (3) corresponds to <code>BitmapTypeV3</code>. Use <code>BmpGetVersion</code> to obtain the contents of this field.</td>
</tr>
</tbody>
</table>

#### Comments

Note the following about the `BitmapType` structures:

- None of these fields contains the actual bitmap data. Instead, the bitmap data is stored immediately following the `BitmapTypeVx` (which one depends on the value of the version field) header structure. If the bitmap has its own color table, the color table is stored in between the header and the data. If the bitmap has a pixel size of 16, and the bitmap is `BitmapTypeV2`, the `BitmapDirectInfoType` structure is stored between the header and the data. You can retrieve a bitmap’s data by passing its `BitmapType` structure to `BmpGetBits`, and you can retrieve its color table with `BmpGetColortable`.

- Unlike most other user interface structures, the `BitmapType` does not store the bitmap’s location on the screen. The `WindowType` or the `FormBitmapType` with which this bitmap is associated contains that information.

- A bitmap may be part of a bitmap family. A bitmap family is a group of bitmaps, each containing the same drawing but at
a different pixel depth (see “Bitmaps” on page 123 of the Palm OS Programmer’s Companion, vol. I). When requested to draw a bitmap family, the operating system chooses a member of the bitmap family based upon the bitmap density and pixel depth; see “Bitmap Families” on page 18 for the algorithm that the High-Density Display Feature Set uses to determine which one to choose.

New **BitmapTypeV0**

Structure corresponding to the version 0 encoding of a bitmap. Version 0 encoding is supported in Palm OS 1.0 and later.

Generally you work with pointers to `BitmapType` structures; if the structure’s version is `BitmapVersionZero`, the structure is of type `BitmapTypeV0`.

**WARNING!** This structure is documented so that you can directly access the internals of your own bitmap resources. *Bitmaps created by Palm OS are not guaranteed to adhere to this structure*; you cannot cast a bitmap created by the Palm OS to this structure and expect to be able to directly access the structure’s fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

```
typedef struct BitmapTypeV0 {
    Int16 width;
    Int16 height;
    UInt16 rowBytes;
    BitmapFlagsType flags;
    UInt16 reserved[4];
} BitmapTypeV0;

typedef BitmapTypeV0 *BitmapPtrV0;
```
Field Descriptions

width
The width of the bitmap in pixels. You specify this value when you create the bitmap. Use BmpGetDimensions to access this field.

height
The height of the bitmap in pixels. You specify this value when you create the bitmap. Use BmpGetDimensions to access this field.

rowBytes
The number of bytes stored for each row of the bitmap where height is the number of rows. Use BmpGetDimensions to access this field.

flags
The bitmap’s attributes. See BitmapFlagsType. Only the compressed flag is defined for BitmapTypeV0 structures.

reserved
Reserved. These values are set to zero. Note that in the BitmapTypeV0 structure, the pixelSize and version fields, defined in BitmapType, do not exist. They coincide with the reserved array, however, and this array was initialized to zero when the bitmap was created. The operating system recognizes that a pixelSize of zero means that the bitmap’s depth is 1.

Compatibility
BitmapTypeV0 is defined only if High-Density Display Feature Set is present.

New

BitmapTypeV1
Structure corresponding to the version 1 encoding of a bitmap. Version 1 encoding is supported in Palm OS 3.0 and later.

Generally you work with pointers to BitmapType structures; if the structure’s version is BitmapVersionOne, the structure is of type BitmapTypeV1.
WARNING!  This structure is documented so that you can directly access the internals of your own bitmap resources. *Bitmaps created by Palm OS are not guaranteed to adhere to this structure*; you cannot cast a bitmap created by the Palm OS to this structure and expect to be able to directly access the structure’s fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

```c
typedef struct BitmapTypeV1 {
    Int16 width;
    Int16 height;
    UInt16 rowBytes;
    BitmapFlagsType flags;
    UInt8 pixelSize;
    UInt8 version;
    UInt16 nextDepthOffset;
    UInt16 reserved[2];
} BitmapTypeV1;

typedef BitmapTypeV1* BitmapPtrV1;
```

**Field Descriptions**

- **width**  
The width of the bitmap in pixels. You specify this value when you create the bitmap. Use `BmpGetDimensions` to access this field.

- **height**  
The height of the bitmap in pixels. You specify this value when you create the bitmap. Use `BmpGetDimensions` to access this field.

- **rowBytes**  
The number of bytes stored for each row of the bitmap where `height` is the number of rows. Use `BmpGetDimensions` to access this field.
**Bitmaps**

**Bitmap Data Structures**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>The bitmap’s attributes. See <a href="#">BitmapFlagsType</a>. Only the compressed and hasColorTable flags are defined for BitmapTypeV1 structures.</td>
</tr>
<tr>
<td>pixelSize</td>
<td>The pixel depth. Currently supported pixel depths are 1, 2, and 4-bit. You specify this value when you create the bitmap. Use BmpGetBitDepth to obtain the contents of this field.</td>
</tr>
<tr>
<td>version</td>
<td>The version of bitmap encoding used. This field has a value of BitmapVersionOne (1) for BitmapTypeV1 structures. Use BmpGetVersion to obtain the contents of this field.</td>
</tr>
<tr>
<td>nextDepthOffset</td>
<td>For bitmap families, this field specifies the start of the next bitmap in the family. The value it contains is the number of 4-byte words to the next BitmapType from the beginning of this one. If the bitmap is not part of a bitmap family or it is the last bitmap in the family, the nextDepthOffset is 0.</td>
</tr>
<tr>
<td>reserved</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

**Compatibility**

BitmapTypeV1 is defined only if [High-Density Display Feature Set](#) is present.

**New**

**BitmapTypeV2**

Structure corresponding to the version 2 encoding of a bitmap. Version 2 encoding is supported in Palm OS 3.5 and later.

Generally you work with pointers to BitmapType structures; if the structure’s version is BitmapVersionTwo (2), the structure is of type BitmapTypeV2.
WARNING! This structure is documented so that you can directly access the internals of bitmaps that you create. Bitmaps created by Palm OS are not guaranteed to adhere to this structure; you cannot cast a bitmap created by the Palm OS to this structure and expect to be able to directly access the structure’s fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

```c
typedef struct BitmapTypeV2 {
    Int16 width;
    Int16 height;
    UInt16 rowBytes;
    BitmapFlagsType flags;
    UInt8 pixelSize;
    UInt8 version;
    UInt16 nextDepthOffset;
    UInt8 transparentIndex;
    UInt8 compressionType;
    UInt16 reserved;
} BitmapTypeV2;

typedef BitmapTypeV2* BitmapPtrV2;
```

Field Descriptions

- **width**: The width of the bitmap in pixels. You specify this value when you create the bitmap. Use `BmpGetDimensions` to access this field.

- **height**: The height of the bitmap in pixels. You specify this value when you create the bitmap. Use `BmpGetDimensions` to access this field.

- **rowBytes**: The number of bytes stored for each row of the bitmap where height is the number of rows. Use `BmpGetDimensions` to access this field.
flags The bitmap’s attributes. See `BitmapFlagsType`. Only the compressed, hasColorTable, hasTransparency, indirect, forScreen, and directColor flags are defined for `BitmapTypeV2` structures. Note that the indirect and forScreen flags are system-only flags that are not used in user-created bitmap resources.

pixelSize The pixel depth. Currently supported pixel depths are 1, 2, 4, 8, and 16-bit. You specify this value when you create the bitmap. Use `BmpGetBitDepth` to obtain the contents of this field.

version The version of bitmap encoding used. This field has a value of `BitmapVersionTwo` (2) for `BitmapTypeV2` structures. Use `BmpGetVersion` to obtain the contents of this field.

nextDepthOffset For bitmap families, this field specifies the start of the next bitmap in the family. The value it contains is the number of 4-byte words to the next `BitmapType` from the beginning of this one. If the bitmap is not part of a bitmap family or it is the last bitmap in the family, the `nextDepthOffset` is 0.

transparentIndex The color index for the transparent color. Only used for version 2 bitmaps and only when the hasTransparency flag is set (see `BitmapFlagsType`). You specify this value when you create the bitmap using Constructor, or programmatically with `BmpSetTransparentValue`. To obtain the value of this field, call `BmpGetTransparentValue`. 

**BitmapTypeV3**

Structure corresponding to the version 3 encoding of a bitmap. Version 3 encoding is supported if the High-Density Display Feature Set is present.

Generally you work with pointers to BitmapType structures; if the structure’s version is BitmapVersionThree (3), the structure is of type BitmapTypeV3.

BmpCreate allocates and initializes a BitmapTypeV2 structure. To create a BitmapTypeV3 structure, use BmpCreateBitmapV3 and supply the data pointer and optional color table pointer.

In earlier versions of the BitmapTypeVx structure, the size of compressed bitmap data is stored in a 16-bit field preceding the bitmap data. With the version 3 structure, the size is stored in a 32-bit field.

The BitmapTypeV3 structure has fields that identify how each pixel is stored (pixelFormat) and which color, if any, is “transparent” (transparentValue). Because of this, you don’t use a BitmapDirectInfoType structure in conjunction with a BitmapTypeV3 structure.
WARNING! This structure is documented so that you can directly access the internals of bitmaps that you create. Bitmaps created by Palm OS are not guaranteed to adhere to this structure; you cannot cast a bitmap created by the Palm OS to this structure and expect to be able to directly access the structure’s fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

```c
typedef struct BitmapTypeV3 {
    Int16 width;
    Int16 height;
    Uint16 rowBytes;
    BitmapFlagsType flags;
    Uint8 pixelSize;
    Uint8 version;
    Uint8 size;
    Uint8 pixelFormat;
    Uint8 unused;
    Uint8 compressionType;
    Uint16 density;
    Uint32 transparentValue;
    Uint32 nextBitmapOffset;
} BitmapTypeV3;

typedef BitmapTypeV3* BitmapPtrV3;
```

Field Descriptions

- **width**: The width of the bitmap in pixels. You specify this value when you create the bitmap. Use `BmpGetDimensions` to access this field.

- **height**: The height of the bitmap in pixels. You specify this value when you create the bitmap. Use `BmpGetDimensions` to access this field.
rowBytes

The number of bytes stored for each row of the bitmap where height is the number of rows. Use `BmpGetDimensions` to access this field.

flags

The bitmap’s attributes. See `BitmapFlagsType`. Note that the indirect, forScreen, and indirectColorTable flags are system-only fields that are not used in user-created bitmap resources.

pixelSize

The pixel depth. Currently supported pixel depths are 1, 2, 4, and 8-bit. You specify this value when you create the bitmap. Use `BmpGetBitDepth` to obtain the value of this field.

version

The version of bitmap encoding used. This field has a value of `BitmapVersionThree` (3) for `BitmapTypeV3` structures. The high bit of the version field is set if the bitmap data structure uses the native ARM format, with little-endian fields. `Bitmap.h` contains a bit mask, `BitmapVersionMaskLE`, that can be used to detect this. Use `BmpGetVersion` to obtain the value of this field.

size

The size of this structure, in bytes. This field does not include the size of the color table or the size of the bitmap data. Use `BmpGetSizes` to obtain the value of this field.

pixelFormat

An enumerated constant representing the format of the pixel data. See `PixelFormatType` for the supported values.

unused

Not used.
**Bitmaps**

*Bitmap Data Structures*

- **compressionType**: The compression type used; 0 if the bitmap is not compressed. Only used when the compressed flag is set (see `BitmapFlagsType`). See `BitmapCompressionType` for possible values. The `BmpCompress` function sets this field, and the `BmpGetCompressionType` function obtains its value.

- **density**: Value used by the blitter to determine how to stretch or shrink the bitmap data. For the screen bitmap, this field represents the screen density. For handhelds with low-density displays, this field is initialized to `kDensityLow`. For handhelds with double-density displays, this field is initialized to `kDensityDouble`. See `DensityType` for the full set of values that this field can assume. Set this field with `BmpSetDensity`, and obtain its value with `BmpGetDensity`.

- **transparentValue**: If this structure represents a bitmap with a bit depth of 8 or less, this field contains the bitmap’s transparent index. If the bitmap has a bit depth of 16, the 16-bit transparent RGB color is stored in this field. You specify this value when you create the bitmap using `Constructor`, or programmatically with `BmpSetTransparentColor`. To obtain the value of this field, call `BmpGetTransparentColor`.

- **nextBitmapOffset**: A 32-bit value that indicates the number of bytes to the next bitmap in the family. If the bitmap is not part of a bitmap family or it is the last bitmap in the family, the `nextBitmapOffset` is 0.

**Compatibility**

BitmapTypeV3 is defined only if the `High-Density Display Feature Set` is present.
**ColorTableType**

The ColorTableType structure defines a color table. Bitmaps can have color tables attached to them; however, doing so is not recommended for performance reasons.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the ColorTableType structure. Never access its structure members directly, or your code may break in future versions. Use BmpGetColortable to access this structure. Use the information below for debugging purposes only.

```c
typedef struct ColorTableType {
   UInt16            numEntries;
    // RGBColorType   entry[];
} ColorTableType;
```

**Field Descriptions**

- **numEntries**: The number of entries in table. High bits (numEntries > 256) reserved.

The color table entries themselves are of type RGBColorType, and there is one per numEntries. Use the macro ColorTableEntries to retrieve these entries.

Care should be taken not to confuse a full color table (which includes the count) with an array of RGB color values. Some routines operate on entire color tables; others operate on lists of color entries.

**Compatibility**

ColorTableType is defined only if 3.5 New Feature Set is present.

---

**New**

**DensityType**

The density of the bitmap (see “Display Density” on page 75 of the Palm OS Programmer’s Companion, vol. I for a definition of display density). Density is only supported in BitmapType structures with
a version greater than 2; if a given BitmapType structure is version 2 or lower, it is assumed to contain low-density data.

The blitter uses the density field in the source and destination bitmaps to determine an appropriate scaling factor. When scaling down from a density of kDensityDouble to kDensityLow, the blitter must shrink the bitmap data. This will almost always result in a poorer-quality image when compared with a bitmap that was created with a density of kDensityLow.

The various DensityType values should not be interpreted as representing pixels per inch.

```c
typedef enum {
    kDensityLow = 72,
    kDensityOneAndAHalf = 108,
    kDensityDouble = 144,
    kDensityTriple = 216,
    kDensityQuadruple = 288
} DensityType
```

**Value Descriptions**

**kDensityLow** Low (single) density. A low-density screen is 160x160 pixels.

**kDensityOneAndAHalf** “One and a half” density. A one-and-a-half-density display is 240x240 pixels; this would most likely be used on a handheld with a 240x320 screen where the bottom portion is used as a “soft Graffiti” area.

**kDensityDouble** Double density when compared with kDensityLow. A double-density screen is 320x320 pixels.

**kDensityTriple** Triple density when compared with kDensityLow. A triple-density screen is 480x480 pixels.

**kDensityQuadruple** Quadruple density when compared with kDensityLow. A quadruple-density screen is 640x640 pixels.
**IMPORTANT:** Not all densities listed in the `DensityType` enum are supported by a given version of the High-Density Display feature set. For Palm OS 5, only `kDensityLow` and `kDensityDouble` are supported.

**Compatibility**

`DensityType` is defined only if [High-Density Display Feature Set](https://example.com) is present.

---

**New**

**PixelFormatType**

Pixel formats defined for use with `BitmapTypeV3` structures.

```c
typedef enum {
    pixelFormatIndexed,
    pixelFormat565,
    pixelFormat565LE,
    pixelFormatIndexedLE
} PixelFormatType;
```

**Field Descriptions**

- **pixelFormatIndexed**: Each pixel is represented by a palette index.
- **pixelFormat565**: Each pixel is represented by an RGB triplet stored in 16-bits: 5 red bits, 6 green bits, and 5 blue bits.
- **pixelFormat565LE**: Similar to `pixelFormat565`, except that the 16 bits of the RGB triplet are stored as little-endian. This pixel format is not supported in user-created bitmaps.
- **pixelFormatIndexedLE**: Similar to `pixelFormatIndexed`, except that the pixels within a byte are stored as little-endian. This pixel format is not supported in user-created bitmaps.
**Compatibl**ity

PixelFormatType is defined only if [High-Density Display Feature Set](#) is present.

**RGBColorType**

The RGBColorType structure defines a color. It is used as an entry in the color table. RGBColorTypes can also be created manually and passed to several user interface functions.

```c
typedef struct RGBColorType {
    UInt8 index;
    UInt8 r;
    UInt8 g;
    UInt8 b;
} RGBColorType;
```

**Field Descriptions**

**index**

The index of this color in the color table. Not all functions that use RGBColorType use the index field.

Direct bitmaps support no more than 256 colors. The number of possible RGB colors greatly exceeds this amount. For this reason, some drawing functions use a color look up table (CLUT). If the CLUT is used, the index field contains the index of an available color that is the closest match to the color specified by the `r`, `g`, and `b` fields.

**r**

Amount of red (0 to 255).

**g**

Amount of green (0 to 255).

**b**

Amount of blue (0 to 255).

**Compatibl**ity

RGBColorType is defined only if [3.5 New Feature Set](#) is present.
Bitmap Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BitmapVersionZero</td>
<td>0</td>
<td>Uses the version 0 encoding of a bitmap. Version 0 encoding is supported in Palm OS 1.0 and later.</td>
</tr>
<tr>
<td>BitmapVersionOne</td>
<td>1</td>
<td>Uses the version 1 encoding of a bitmap. Version 1 encoding is supported in Palm OS 3.0 and later.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PalmRez automatically creates version 1 bitmaps unless you have specified a transparency index or a compressed type when creating the bitmap in Constructor.</td>
</tr>
<tr>
<td>BitmapVersionTwo</td>
<td>2</td>
<td>Uses the version 2 encoding of a bitmap. Palm OS 3.5 and later supports version 2 bitmaps. Version 2 bitmaps either use the transparency index or are compressed. If you programmatically create a bitmap using BmpCreate, a version 2 bitmap is created.</td>
</tr>
<tr>
<td>BitmapVersionThree</td>
<td>3</td>
<td>Uses the version 3 encoding of a bitmap. Version 3 bitmaps are supported only if the High-Density Display Feature Set is present.</td>
</tr>
</tbody>
</table>

Bitmap Resources

You can create a bitmap resource and include it as part of your application’s PRC file. Use the resource type 'Tbmp' for most images and the resource type 'tAIB' for application icons. Symbolically, these two resource types are bitmapRsc and iconType, respectively.

Note that if you are creating a bitmap or a bitmap family in Constructor, you create a 'tbmf' resource (or 'taif' resource for icons) and one or more 'PICT' images. The PalmRez post linker converts them into a single 'Tbmp' or 'tAIB' resource. Note that
the PalmRez post linker takes PICT images even on the Microsoft Windows operating system.

**Bitmap Functions**

**BmpBitsSize**

**Purpose**
Return the size of the bitmap’s data.

**Declared In**
Bitmap.h

**Prototype**
UInt16 BmpBitsSize (const BitmapType *bitmapP)

**Parameters**
-> bitmapP Pointer to the bitmap. (See BitmapType.)

**Result**
Returns the size in bytes of the bitmap’s data, excluding the header and the color table.

**Comments**
This function returns the bitmap’s data size even if the bitmap’s indirect flag is set. (See BitmapFlagsType.)
If the bitmap is compressed, this function returns the compressed size of the bitmap.

**Compatibility**
Implemented only if 3.5 New Feature Set is present.

**See Also**
BmpSize, BmpColortableSize, BmpGetBits
BmpColortableSize

**Purpose**
Return the size of the bitmap’s color table.

**Declared In**
Bitmap.h

**Prototype**
```c
UInt16 BmpColortableSize
(const BitmapType *bitmapP)
```

**Parameters**
- `bitmapP` Pointer to the bitmap. (See `BitmapType`.)

**Result**
Returns the size in bytes of the bitmap’s color table or 0 if the bitmap does not use its own color table.

**Compatibility**
Implemented only if 3.5 New Feature Set is present.

**See Also**
BmpBitsSize, BmpSize, BmpGetColortable

BmpCompress

**Purpose**
Compress or uncompress a bitmap.

**Declared In**
Bitmap.h

**Prototype**
```c
Err BmpCompress (BitmapType *bitmapP,
BitmapCompressionType compType)
```

**Parameters**
- `bitmapP` Pointer to the bitmap to compress. (See `BitmapType`.)
- `compType` The type of compression to use. (See `BitmapCompressionType`.) If set to `BitmapCompressionTypeNone` and `bitmapP` is compressed, this function uncompresses the bitmap.

**Result**
Returns one of the following values:
- `errNone` Success.
sysErrParamErr  Either the compType parameter does not specify a compression type or the bitmap is already compressed, is in the storage heap, or represents the screen.

sysErrNoFreeResource  There is not enough memory available to complete the operation.

Comments  This function performs the specified compression and resizes the bitmap’s allocated memory. The bitmap must be in the dynamic heap.

Compatibility  Implemented only if 3.5 New Feature Set is present.

See Also  BmpGetCompressionType

BmpCreate

Purpose  Create a bitmap.

Declared In  Bitmap.h

Prototype  BitmapType *BmpCreate (Coord width, Coord height, UInt8 depth, ColorTableType *colortableP, UInt16 *error)

Parameters  

-> width  The width of the bitmap in pixels. Must not be 0.

-> height  The height of the bitmap in pixels. Must not be 0.

-> depth  The pixel depth of the bitmap. Must be 1, 2, 4, 8, or 16. This value is used as the pixelSize field of BitmapType.
-> colortableP A pointer to the color table associated with the bitmap, or NULL if the bitmap should not include a color table. If specified, the number of colors in the color table must match the depth parameter. (2 for 1-bit, 4 for 2-bit, 16 for 4-bit, and 256 for 8-bit). 16-bit bitmaps do not use a color table.

<- error Contains the error code if an error occurs.

**Result** Returns a pointer to the new bitmap structure (see BitmapType) or NULL if an error occurs. The parameter error contains one of the following:

- errNone Success.
- sysErrParamErr The width, height, depth, or colorTableP parameter is invalid. See the descriptions above for acceptable values.
- sysErrNoFreeResource There is not enough memory available to allocate the structure.

**Comments** This function creates an uncompressed, non-transparent BitmapVersionTwo bitmap with the width, height, and depth that you specify. To create a BitmapVersionThree bitmap use BmpCreate and pass the results to BmpCreateBitmapV3.

If you pass a color table, the bitmap’s hasColorTable flag is set. For performance reasons, attaching a custom color table to a bitmap is strongly discouraged. An alternative is to use the WinPalette command to change the color table as needed, draw the bitmap, and then undo your changes after you have finished displaying the bitmap.

BmpCreate allocates sufficient memory on the dynamic heap to hold the bitmap and initializes all of its pixels to white. To change the bitmap’s contents, use the window drawing functions. First, you must use WinCreateBitmapWindow to create an off screen window wrapper around the bitmap, then draw to that window. For example:
Bitmaps

Blobmap Functions

BitmapType *bmpP;
WinHandle win;
Err error;
RectangleType onScreenRect;

bmpP = BmpCreate(10, 10, 8, NULL, &error);
if (bmpP) {
    win = WinCreateBitmapWindow(bmpP, &error);
    if (win) {
        WinSetDrawWindow(win);
        WinDrawLines(win, ...);
        /* etc */
        WinSetWindowBounds(win, onScreenRect);
    }
}

You cannot use this function to create a bitmap written directly to a
database; that is, you must create the bitmap on the dynamic heap
first, then write it to the storage heap.

It is not necessary to use BmpCreate to load a bitmap stored in a
resource. Instead, you simply load the resource and lock its handle.
The returned pointer is a pointer to a BitmapType. For example:

MemHandle resH =
    DmGetResource (bitmapRsc, rscID);
BitmapType *bitmap = MemHandleLock (resH);

Bitmaps 64 Kb and greater are now supported with Palm OS 4.0.

Compatibility  Implemented only if 3.5 New Feature Set is present.

See Also  BmpCreateBitmapV3, BmpDelete
**New**

**BmpCreateBitmapV3**

**Purpose**
Create a version 3 bitmap from an existing bitmap, an existing set of data bits, and, optionally, a color table.

**Declared In**
Bitmap.h

**Prototype**
```c
BitmapTypeV3 *BmpCreateBitmapV3
(const BitmapType *bitmapP, UInt16 density,
 const void *bitsP,
 const ColorTableType *colorTableP)
```

**Parameters**
- `-> bitmapP` Pointer to a valid bitmap from which the version 3 bitmap is to be created. See BitmapType.
- `-> density` Density of the returned bitmap. If 0, the returned bitmap’s density is set to the default value of kDensityLow.
- `-> bitsP` Pointer to the bitmap image data. Note that the bitmap data can be located in the storage heap, but then the bitmap should be treated as read-only. You must use DmWrite to write to the storage heap; blitting to it causes a system error.
- `-> colorTableP` Pointer to a color table, or NULL to use bitmapP’s color table, if one exists.

**Result**
Returns a version 3 bitmap, or NULL if the bitmap could not be created from the specified bitmap, bitmap data, and optional color table.

**Comments**
You can use this function when the bitmap data is stored in the storage heap as bands of raster data. Rather than allocating several bitmap structures, one for each band, use this function to allocate a single bitmap, and have the structure point to each band successively. This is typically used with high-density bitmaps that cannot be stored entirely within 64k.
WARNING! Due to a limitation in the way that this function is implemented, BitmapCreateBitmapV3 doesn't work with compressed bitmaps. Don't pass bitmaps to this function that have the compressed flag set.

The returned bitmap structure is allocated from the system heap. After your application is done with it, dispose of it by calling BmpDelete.

Compatibility Implemented only if the High-Density Display Feature Set is present.

See Also BmpCreate

BmpDelete

Purpose Delete a bitmap structure.

Declared In Bitmap.h

Prototype Err BmpDelete (BitmapType *bitmapP)

Parameters ->bitmapP Pointer to the structure of the bitmap to be deleted. (See BitmapType.)

Result Returns errNone upon success, sysErrParamErr if the bitmap’s forScreen flag is set or the bitmap resides in the storage heap. Returns one of the memory errors if the freeing pointer fails.

Comments Only delete bitmaps that have been created using BmpCreate. You cannot use this function on a bitmap located in a database. To delete a bitmap from a database, use the standard data manager calls.

Compatibility Implemented only if 3.5 New Feature Set is present.
BmpGetBits

**Purpose** Retrieve the bitmap’s data.

**Declared In** Bitmap.h

**Prototype** void *BmpGetBits (BitmapType *bitmapP)

**Parameters**
- `-> bitmapP` Pointer to the bitmap’s structure. (See BitmapType.)

**Result** Returns a pointer to the bitmap’s data.

**Comments** This function returns the bitmap’s data even if the bitmap’s indirect flag is set. (See BitmapFlagsType.)

**Compatibility** Implemented only if 3.5 New Feature Set is present.

**See Also** BmpBitsSize

BmpGetBitDepth

**Purpose** Retrieve the depth of a bitmap.

**Declared In** Bitmap.h

**Prototype** UInt8 BmpGetBitDepth (const BitmapType* bitmapP)

**Parameters**
- `-> bitmapP` Pointer to a bitmap. See BitmapType.

**Result** This function returns the bit depth of the bitmap, as represented by the pixelSize field in BitmapType. For debug ROMs, this function reports an error and returns 0 if bitmapP is NULL.

**Compatibility** Implemented only if 4.0 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS,
link with the PalmOSGlue library and call BmpGlueGetBitDepth. For more information, see Chapter 75, "PalmOSGlue Library."

See Also BmpGetDimensions, BmpGetNextBitmap, BmpGetSizes

**BmpGetColortable**

**Purpose** Retrieve the bitmap’s color table.

**Declared In** Bitmap.h

**Prototype** 
```
ColorTableType *BmpGetColortable
(BitmapType *bitmapP)
```

**Parameters** 
- **bitmapP** A pointer to the bitmap. See BitmapType.

**Result** Returns a pointer to the color table or NULL if the bitmap uses the system color table.

**Compatibility** Implemented only if 3.5 New Feature Set is present.

See Also BmpColortableSize

**BmpGetCompressionType**

**Purpose** Get the compression type of a bitmap.

**Declared In** Bitmap.h

**Prototype** 
```
BitmapCompressionType BmpGetCompressionType
(const BitmapType *bitmapP)
```

**Parameters** 
- **bitmapP** Pointer to a valid bitmap. See BitmapType.

**Result** Returns the type of compression used by bitmapP. See “BitmapCompressionType” on page 513 for the values that can be
returned from this function. For debug ROMs, this function reports an error and returns BitmapCompressionTypeNone if bitmapP is NULL.

**Comments**

If the bitmap is not compressed, this function returns BitmapCompressionTypeNone. If the bitmap version is 0 or 1 (corresponding to BitmapTypeV0 and BitmapTypeV1, respectively), it returns BitmapCompressionTypeScanLine.

**Compatibility**

Implemented only if either the if 5.0 New Feature Set or the High-Density Display Feature Set is present.

**See Also**

BmpCompress

---

**New**

**BmpGetDensity**

**Purpose**

Get the density of a bitmap.

**Declared In**

Bitmap.h

**Prototype**

`UInt16 BmpGetDensity (const BitmapType *bitmapP)`

**Parameters**

-> bitmapP Pointer to a valid bitmap. See BitmapType.

**Result**

Returns the density of bitmapP; see the `DensityType` enum for the defined set of density values. For debug ROMs, this function reports an error and returns 0 if bitmapP is NULL.

**Comments**

Note that bitmaps with a version of 0, 1, or 2 (corresponding to BitmapTypeV0, BitmapTypeV1, and BitmapTypeV2, respectively) are assumed to be low density (kDensityLow).

**Compatibility**

Implemented only if the High-Density Display Feature Set is present.

**See Also**

BmpCreateBitmapV3, BmpSetDensity
**BmpGetDimensions**

**Purpose**
Retrieve the width, height and number of data bytes per row of a bitmap.

**Declared In**
Bitmap.h

**Prototype**
```c
void BmpGetDimensions (const BitmapType *bitmapP, Coord *widthP, Coord *heightP, UInt16 *rowBytesP)
```

**Parameters**
- `*bitmapP` Pointer to the bitmap. See `BitmapType`.
- `*widthP` Pointer to bitmap’s width in pixels. Use `NULL` if this information is not wanted.
- `*heightP` Pointer to bitmap’s height in pixels. Use `NULL` if this information is not wanted.
- `*rowBytesP` Pointer to number of bytes per row of bitmap. Use `NULL` if this information is not wanted.

**Result**
This function returns the width in pixels of the bitmap in `widthP`, the height in pixels of the bitmap in `heightP`, and the number of bytes of data per row of the bitmap in `rowBytesP`. This function reports an error on debug ROMs if `bitmapP` is `NULL`.

**Compatibility**
Implemented only if [4.0 New Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `BmpGlueGetDimensions`. For more information, see Chapter 75, "PalmOSGlue Library."

**See Also**
- `BmpGetBitDepth`
- `BmpGetNextBitmap`
- `BmpGetSizes`
**BmpGetNextBitmap**

**Purpose**  Retrieve the next low-density bitmap in a bitmap family.

**Declared In**  Bitmap.h

**Prototype**  

```c
BitmapType *BmpGetNextBitmap(BitmapType *bitmapP)
```

**Parameters**  

- `bitmapP`  Pointer to a bitmap. See `BitmapType`.

**Result**  This function returns a pointer to the next low-density `BitmapType` in a bitmap family. It returns `NULL` if `bitmapP` is the last bitmap. For debug ROMs, this function reports an error and returns 0 if `bitmapP` is `NULL`.

**Compatibility**  Implemented only if 4.0 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `BmpGlueGetNextBitmap`. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**  `BmpGetBitDepth`, `BmpGetDimensions`, `BmpGetNextBitmapAnyDensity`, `BmpGetSizes`
New **BmpGetNextBitmapAnyDensity**

**Purpose**
Get the next bitmap in the bitmap family, irrespective of density.

**Declared In**
Bitmap.h

**Prototype**
```
BitmapType *BmpGetNextBitmapAnyDensity
(BitmapType *bitmapP)
```

**Parameters**
- `-> bitmapP` Pointer to a valid bitmap. See [BitmapType](#).

**Result**
Returns the next bitmap in a bitmap family, or NULL if `bitmapP` is
the last bitmap. For debug ROMs, this function reports an error and
returns 0 if `bitmapP` is NULL.

**Comments**
This function is an extended version of [BmpGetNextBitmap](#). For
backward compatibility, BmpGetNextBitmap only returns low-density
bitmaps. If the bitmap family contains high-density
bitmaps, however, BmpGetNextBitmapAnyDensity skips
over the dummy bitmap that separates the low and high-density
bitmaps in the linked list and returns a high-density bitmap.

**Compatibility**
Implemented only if the [High-Density Display Feature Set](#) is
present.

**See Also**
[BmpGetDensity](#), [BmpGetNextBitmap](#)
BmpGetSizes

PurposeRetrieve the size of a bitmap and its header structure.

Declared In Bitmap.h

Prototype

```c
void BmpGetSizes (const BitmapType *bitmapP, UInt32 *dataSizeP, UInt32 *headerSizeP)
```

Parameters

- **bitmapP** Pointer to the bitmap. See BitmapType.
- **dataSizeP** Pointer to size of bitmap data, not including structures. Use NULL if this information is not wanted.
- **headerSizeP** Pointer to size of bitmap’s structures, not including data. Use NULL if this information is not wanted.

Result

Returns the size of the bitmap and the size of the bitmap’s structures. This function will report an error on debug ROMs if bitmapP is NULL.

Comments

This function returns the size in bytes of the bitmap data in dataSizeP. The size does not include the data structures (BitmapType, BitmapDirectInfoType, or color table) that are associated with a bitmap. The size of the structures (in bytes) are returned in headerSizeP, which includes the size of the BitmapType, BitmapDirectInfoType (if any), the color table (if any), and the size of the pointer for indirect bitmaps (described in BitmapFlagsType).

This function should be used when working with bitmaps that may be 64 Kb or greater. Do not use BmpSize or BmpBitsSize when working with bitmaps that may be greater than or equal to 64Kb.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

BmpGetBitDepth, BmpGetDimensions, BmpGetNextBitmap
### BmpGetTransparentColorValue

**Purpose**
Get a bitmap’s transparent color.

**Declared In**
Bitmap.h

**Prototype**
```c
Boolean BmpGetTransparentColorValue
(const BitmapType *bitmapP,
 UInt32 *transparentValueP)
```

**Parameters**
- `bitmapP` Pointer to a valid bitmap. See `BitmapType`.
- `transparentValueP` Pointer to a variable that receives the transparent color, either as a palette index or as a direct color value (an `RGBColorType`), depending on the bitmap’s depth.

**Result**
Returns `true` if `bitmapP` has a transparent color defined, `false` otherwise.

**Compatibility**
Implemented only if either the if 5.0 New Feature Set or the High-Density Display Feature Set is present.

**See Also**
- `BmpSetTransparentColorValue`
New **BmpGetVersion**

**Purpose**
Get the version of a bitmap.

**Declared In**
Bitmap.h

**Prototype**
`UInt8 BmpGetVersion (const BitmapType *bitmapP)`

**Parameters**
- `bitmapP` Pointer to a valid bitmap. See **BitmapType**.

**Result**
Returns the version of `bitmapP`. See “**Bitmap Constants**” on page 535 for the defined bitmap version numbers. For debug ROMs, this function reports an error and returns 0 if `bitmapP` is NULL.

**Compatibility**
Implemented only if the **High-Density Display Feature Set** is present.

New **BmpSetDensity**

**Purpose**
Set the density of a version 3 bitmap.

**Declared In**
Bitmap.h

**Prototype**
`Err BmpSetDensity (BitmapType *bitmapP, UInt16 density)`

**Parameters**
- `bitmapP` Pointer to a valid version 3 bitmap. See **BitmapTypeV3**.
-> density  The bitmap’s density. This value should be one of the values defined by the `DensityType` enum.

**Result**  Returns `errNone` if the operation completed successfully, or `SysErrParamErr` either if `bitmapP` is `NULL`, if `density` is not supported by the blitter, or if `*bitmapP` is not a version 3 bitmap.

**Comments**  To allocate a high-density bitmap, first call `BmpCreateBitmapV3`. Then call `BmpSetDensity` to specify the bitmap’s density.

**Compatibility**  Implemented only if the High-Density Display Feature Set is present.

**See Also**  `BmpGetDensity`

---

**New**  `BmpSetTransparentColor`

**Purpose**  Set a bitmap’s transparent color.

**Declared In**  `Bitmap.h`

**Prototype**  ```c
void BmpSetTransparentColor (BitmapType *bitmapP, UInt32 transparentValue)
```

**Parameters**

- `-> bitmapP`  Pointer to a valid bitmap. See `BitmapType`.
- `-> transparentValue`  Transparent color. This should either be a palette index or a direct color value (an `RGBColorType`), depending on the bitmap’s depth.

**Result**  Returns nothing.

**Comments**  If `bitmapP` points to a version 2 bitmap, `BmpSetTransparentColor` sets the `BitmapTypeV2` structure’s
hasTransparency flag to true and initializes the structure’s transparentIndex field according to transparentValue. For 16-bit bitmaps, this function sets the transparentColor field in the BitmapDirectInfoType auxiliary structure and sets the transparentIndex field to 0.

If bitmapP points to a version 3 bitmap, BmpSetTransparentValue sets the BitmapTypeV3 structure’s hasTransparency flag to true and sets the transparentValue field to the transparent color.

Regardless of the bitmap version, if this function is passed a transparentValue set to kTransparencyNone, this function sets the bitmap structure’s hasTransparency flag to false and sets the transparent color field(s) to 0.

This function does nothing if transparentValue contains a value that is not valid for the depth of bitmapP.

Compatibility Implemented only if either the if 5.0 New Feature Set or the High-Density Display Feature Set is present.

See Also BmpGetTransparentValue

BmpSize

Purpose Return the size of the bitmap.

Declared In Bitmap.h

Prototype UInt16 BmpSize (const BitmapType *bitmapP)

Parameters -> bitmapP A pointer to the bitmap. See BitmapType.

Result Returns the size in bytes of the bitmap, including its header, color table (if any), and sizeof (BitmapDirectInfoType) if one exists.

Comments If the bitmap has its indirect flag set (see BitmapFlagsType), the bitmap data is not included in the size returned by this function.
**ColorTableEntries**

**Purpose**
Macro that returns the color table.

**Declared In**
Bitmap.h

**Prototype**
ColorTableEntries (ctP)

**Parameters**
- ctP 
  A pointer to a ColorTableType structure.

**Result**
Returns an array of RGBColorType structures, one for each entry in the color table.

**Comments**
You can use this macro to retrieve the RGB values in use by a bitmap. For example:

```c
BitMapType *bmpP;
RGBColorType *tableP =
    ColorTableEntries(BmpGetColorTable(bmpP));
```

If you want to retrieve the RGB values in use by the system color table, you can simply use the WinPalette function instead of this macro:

```c
RGBColorType table[256];
Err e;

/* allocate space for table */
e = WinPalette(winPaletteGet, 0, 256, tableP);
```

**Compatibility**
Implemented only if 3.5 New Feature Set is present.

**See Also**
BmpGetColortable
Character Attributes

This chapter provides reference material for character attributes functions defined in CharAttr.h.

Character Attribute Functions

ChrHorizEllipsis

**Purpose**
Macro that returns the appropriate character code for the horizontal ellipsis.

**Declared In**
Chars.h

**Prototype**
ChrHorizEllipsis (chP)

**Parameters**
<- chP

**Parameters**
Pointer to a variable in which to return the horizontal ellipsis character code.

**Result**
Returns nothing. Upon return, the variable pointed to by chP contains the horizontal ellipsis character.

**Comments**
Version 3.1 of the Palm OS® uses different character codes for the horizontal ellipsis character and the numeric space character than earlier versions did. Use this macro to return the appropriate code for horizontal ellipsis regardless of which version of Palm OS your application is run on.
**ChrIsHardKey**

**Purpose**  
Macro that returns true if the character is one of the hard keys on the device.

**Declared In**  
Chars.h

**Prototype**  
ChrIsHardKey (ch)

**Parameters**  
- ch  
The character from the keyDownEvent.

**Result**  
true if the character is one of the four built-in hard keys on the device, false otherwise.

**Compatibility**  
This macro is obsolete and replaced by TxtCharIsHardKey if the International Feature Set is present.

---

**ChrNumericSpace**

**Purpose**  
Macro that returns the appropriate character code for the numeric space.

**Declared In**  
Chars.h

**Prototype**  
ChrNumericSpace (chP)

**Parameters**  
- chP  
Pointer to a variable in which to return the numeric space character code.

**Result**  
Returns nothing. Upon return, the variable pointed to by chP contains the numeric space character.

**Comments**  
Version 3.1 of the Palm OS uses different character codes for the horizontal ellipsis character and the numeric space character than earlier versions did. Use this macro to return the appropriate code for numeric space regardless of which version of Palm OS your application is run on.
GetCharAttr

**Purpose**  Return a pointer to the character attribute array. This array is used by the character classification and character conversion macros (such as `isalpha`).

**Declared In**  CharAttr.h

**Prototype**  `UInt16* GetCharAttr (void)`

**Parameters**  None

**Result**  A pointer to the attributes array. This is an array of 256 `UInt16` values, one for each possible character code. See CharAttr.h for an explanation of the attributes.

**Compatibility**  This function is not implemented if International Feature Set is present.

---

**NOTE:** This function is provided for backwards compatibility only. Use Text Manager functions instead on systems that support the text manager.

If 5.0 New Feature Set is present, this function is implemented in PACE. However, it only supports the Latin table, regardless of the localization.

**See Also**  TxtCharAttr, TxtCharXAttr
GetCharCaselessValue

**Purpose**
Return a pointer to an array that maps all characters to an assigned caseless and accentless value. Use this function for finding text.

**Declared In**
CharAttr.h

**Prototype**
UInt8* GetCharCaselessValue (void)

**Parameters**
None.

**Result**
Returns a pointer to the sort array, which is an array of 256 bytes.

**Comment**
The GetCharCaselessValue conversion table converts each character into a numeric value that is caseless and sorted according to Microsoft Windows sorting rules:
- Punctuation characters have the lowest values,
- followed by numbers,
- followed by alpha characters.
  
  All forms of each alpha character have equivalent values, so that e = E = e-grave = e-circumflex, etc.

This conversion table is used by all the Palm OS sorting and comparison routines to yield caseless searches and caseless sorts in the almost same order as Windows-based programs, except that Palm OS routines produce the same sorting for all locales.

**Compatibility**
This function is not implemented if International Feature Set is present.

NOTE: This function is provided for backwards compatibility only. Use Text Manager functions instead on systems that support the text manager.

If 5.0 New Feature Set is present, this function is implemented in PACE. However, it only supports the Latin table, regardless of the localization.
GetCharSortValue

**Purpose**
Return a pointer to an array that maps all characters to an assigned sorting value. Use this function for ordering (sorting) text.

**Declared In**
CharAttr.h

**Prototype**
UInt8* GetCharSortValue (void)

**Parameters**
None.

**Result**
Returns a pointer to the attributes array. This is an array of 256 UInt8 values, one for each possible character code.

**Compatibility**
This function is **not** implemented if International Feature Set is present.

---

**NOTE:** This function is provided for backwards compatibility only. Use Text Manager functions instead on systems that support the text manager.

If 5.0 New Feature Set is present, this function is implemented in PACE. However, it only supports the Latin table, regardless of the localization.
Data and Resource Manager

This chapter describes the data manager and the resource manager API declared in the header file DataMgr.h. It discusses the following topics:

- Data Manager Data Structures
- Data Manager Constants
- Data Manager Functions
- Application-Defined Functions

For more information on the data and resource managers, see the chapter “Files and Databases” in the Palm OS Programmer’s Companion, vol. I.

Data Manager Data Structures

**DmOpenRef**

The DmOpenRef type defines a pointer to an open database. The database pointer is created and returned by DmOpenDatabase. It is used in any function that requires access to an open database.

```c
typedef void *DmOpenRef
```

**DmResID**

The DmResID type defines a resource identifier. You assign each resource an ID at creation time. Note that resource IDs greater than or equal to 10000 are reserved for system use.
typedef UInt16 DmResID;

**DmResType**
The DmResType type defines the type of a resource. The resource type is a four-character code such as 'Tbmp' for bitmap resources.

typedef UInt32 DmResType;

**SortRecordInfoType**
The SortRecordInfoType structure specifies information that can be used to sort a record. The database sorting functions (DmInsertionSort and DmQuickSort) pass this structure to your comparison callback function (of type DmComparF), where you can use the information therein to help when comparing two records. To create this structure, you can call DmRecordInfo, which returns these values for a given record.

typedef struct {
    UInt8 attributes;
    UInt8 uniqueID[3];
} SortRecordInfoType;

typedef SortRecordInfoType *SortRecordInfoPtr;

**Field Descriptions**
- **attributes**: The record’s attributes. See “Record Attribute Constants.”
- **uniqueID**: The unique identifier for the record.

**Data Manager Constants**

**Category Constants**
The following constants are used to specify information about categories:
Data and Resource Manager

Data Manager Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmAllCategories</td>
<td>0xFF</td>
<td>A mask used to represent all categories.</td>
</tr>
<tr>
<td>dmCategoryLength</td>
<td>16</td>
<td>The length of a category name. Currently, this is 16 bytes, which includes the null terminator.</td>
</tr>
<tr>
<td>dmRecAttrCategoryMask</td>
<td>0x0F</td>
<td>A mask used to retrieve the category information from the record’s attributes field.</td>
</tr>
<tr>
<td>dmRecNumCategories</td>
<td>16</td>
<td>The number of categories allowed. Currently, this is 16, which includes the “Unfiled” category.</td>
</tr>
<tr>
<td>dmUnfiledCategory</td>
<td>0</td>
<td>A mask used to indicate the Unfiled category.</td>
</tr>
</tbody>
</table>

Record Attribute Constants

The following constants specify a database record’s attributes.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmMaxRecordIndex</td>
<td>0xFFFF</td>
<td>Indicates the highest record index allowed.</td>
</tr>
<tr>
<td>dmAllRecAttrs</td>
<td>0xF0</td>
<td>A mask used to specify all record attributes.</td>
</tr>
<tr>
<td>dmRecAttrBusy</td>
<td>0x20</td>
<td>Busy (the application has locked access to this record). A call to DmGetRecord fails on a record that has this bit set, otherwise it sets this bit. Call DmReleaseRecord to release the record and clear this bit. The DmSetRecordInfo function cannot be used to alter the state of dmRecAttrBusy.</td>
</tr>
<tr>
<td>dmRecAttrDelete</td>
<td>0x80</td>
<td>Deleted</td>
</tr>
<tr>
<td>dmRecAttrDirty</td>
<td>0x40</td>
<td>Dirty (has been modified since last sync)</td>
</tr>
<tr>
<td>dmRecAttrSecret</td>
<td>0x10</td>
<td>Private</td>
</tr>
<tr>
<td>dmSysOnlyRecAttrs</td>
<td>0x20</td>
<td>A mask used to specify record attributes that only the system can change. (In other words, the busy attribute.)</td>
</tr>
</tbody>
</table>
## Database Attribute Constants

The following constants define a database’s attributes:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmAllHdrAttrs</td>
<td>A mask used to specify all header attributes.</td>
</tr>
<tr>
<td>dmHdrAttrAppInfoDirty</td>
<td>The application info block is dirty (has been modified since the last sync).</td>
</tr>
<tr>
<td>dmHdrAttrBackup</td>
<td>The database should be backed up to the desktop computer if no application-specific conduit is available.</td>
</tr>
<tr>
<td>dmHdrAttrBundle</td>
<td>The database is bundled with its application during a beam. That is, if the user chooses to beam the application from the Launcher, the Launcher beams this database along with the application’s resource database and overlay database. This attribute applies to Palm OS® 4.0 and higher. Note that overlay databases are automatically beamed with the application database on Palm OS 4.0 and higher. You do not need to set this bit in overlay databases.</td>
</tr>
<tr>
<td>dmHdrAttrCopyPrevention</td>
<td>Prevents the database from being copied by methods such as IR beaming.</td>
</tr>
<tr>
<td>dmHdrAttrHidden</td>
<td>This database should be hidden from view. For example, this attribute is set to hide some applications in the launcher’s main view. You can set it on record databases to have the launcher disregard the database’s records when showing a count of records. This attribute applies to Palm OS version 3.2 and higher.</td>
</tr>
</tbody>
</table>
The following constants define error codes that are returned by the data manager and resource manager functions. Several functions return a failure value such as `NULL` or 0 instead of an error code. In many cases, you can call `DmGetLastError` upon receiving this value and receive a more descriptive error code.
Also, note that on releases prior to Palm OS release 3.5, many data manager functions display a fatal error message using the `ErrFatalDisplayIf` macro if certain error conditions are true. Because the Palm OS ROMs are usually shipped with error checking set to partial, you receive the fatal error message. If a ROM is built with error checking set to none, the function returns one of the error codes listed here. (Note that Palm™ has never released a ROM with error checking set to none and has no plans to do so.)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dmErrAlreadyExists</code></td>
<td>Another database with the same name already exists in RAM store.</td>
</tr>
<tr>
<td><code>dmErrAlreadyOpenForWrites</code></td>
<td>The database is already open with write access.</td>
</tr>
<tr>
<td><code>dmErrCantFind</code></td>
<td>The specified resource can't be found.</td>
</tr>
<tr>
<td><code>dmErrCantOpen</code></td>
<td>The database cannot be opened.</td>
</tr>
<tr>
<td><code>dmErrCorruptDatabase</code></td>
<td>The database is corrupted.</td>
</tr>
<tr>
<td><code>dmErrDatabaseOpen</code></td>
<td>The function cannot be performed on an open database, and the database is open.</td>
</tr>
<tr>
<td><code>dmErrDatabaseNotProtected</code></td>
<td>DmDatabaseProtect failed to protect the specified database.</td>
</tr>
<tr>
<td><code>dmErrIndexOutOfRange</code></td>
<td>The specified index is out of range.</td>
</tr>
<tr>
<td><code>dmErrInvalidDatabaseName</code></td>
<td>The name you’ve specified for the database is invalid.</td>
</tr>
<tr>
<td><code>dmErrInvalidParam</code></td>
<td>The function received an invalid parameter.</td>
</tr>
<tr>
<td><code>dmErrMemError</code></td>
<td>A memory error occurred.</td>
</tr>
</tbody>
</table>
## Data Manager Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmErrNoOpenDatabase</td>
<td>The function is to search all open databases, but there are none.</td>
</tr>
<tr>
<td>dmErrNotRecordDB</td>
<td>You’ve attempted to perform a record function on a resource database.</td>
</tr>
<tr>
<td>dmErrNotResourceDB</td>
<td>You’ve attempted to perform a resource manager function on a record database.</td>
</tr>
<tr>
<td>dmErrNotValidRecord</td>
<td>The record handle is invalid.</td>
</tr>
<tr>
<td>dmErrOpenedByAnotherTask</td>
<td>You’ve attempted to open a database that another task already has open.</td>
</tr>
<tr>
<td>dmErrReadOnly</td>
<td>You’ve attempted to write to or modify a database that is in read-only mode.</td>
</tr>
<tr>
<td>dmErrRecordArchived</td>
<td>The function requires that the record not be archived, but it is.</td>
</tr>
<tr>
<td>dmErrRecordBusy</td>
<td>The function requires that the record not be busy, but it is.</td>
</tr>
<tr>
<td>dmErrRecordDeleted</td>
<td>The record has been deleted.</td>
</tr>
<tr>
<td>dmErrRecordInWrongCard</td>
<td>You’ve attempted to attach a record to a database when the record and database reside on different memory cards.</td>
</tr>
<tr>
<td>Constant</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>dmErrResourceNotFound</td>
<td>The resource can’t be found.</td>
</tr>
<tr>
<td>dmErrROMBased</td>
<td>You’ve attempted to delete or modify a ROM-based database.</td>
</tr>
<tr>
<td>dmErrSeekFailed</td>
<td>The operation of seeking the next record in the category failed.</td>
</tr>
<tr>
<td>dmErrUniqueIDNotFound</td>
<td>A record with the specified unique ID can’t be found.</td>
</tr>
<tr>
<td>dmErrWriteOutOfBounds</td>
<td>A write operation exceeded the bounds of the record.</td>
</tr>
<tr>
<td>memErrCardNotPresent</td>
<td>The specified card can’t be found.</td>
</tr>
<tr>
<td>memErrChunkLocked</td>
<td>The associated memory chunk is locked.</td>
</tr>
<tr>
<td>memErrInvalidParam</td>
<td>A memory error occurred.</td>
</tr>
<tr>
<td>memErrNotEnoughSpace</td>
<td>A memory error occurred.</td>
</tr>
<tr>
<td>memErrInvalidStoreHeader</td>
<td>The specified card has no storage RAM.</td>
</tr>
<tr>
<td>memErrRAMOnlyCard</td>
<td>An attempt was made to open a stripped resource database, but no associated overlay could be found.</td>
</tr>
<tr>
<td>omErrBaseRequiresOverlay</td>
<td>An attempt was made to open a resource database with overlays using an unknown locale.</td>
</tr>
</tbody>
</table>
Open Mode Constants

The following constants define the mode in which a database can be opened. You pass one or more of these as a parameter to
\texttt{DmOpenDatabase}, \texttt{DmOpenDatabaseByTypeCreator}, or \texttt{DmOpenDBNoOverlay}:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{dmModeReadWrite}</td>
<td>Read-write access.</td>
</tr>
<tr>
<td>\texttt{dmModeReadOnly}</td>
<td>Read-only access.</td>
</tr>
<tr>
<td>\texttt{dmModeWrite}</td>
<td>Write-only access.</td>
</tr>
<tr>
<td>\texttt{dmModeLeaveOpen}</td>
<td>Leave database open even after application quits.</td>
</tr>
<tr>
<td>\texttt{dmModeExclusive}</td>
<td>Don't let anyone else open this database.</td>
</tr>
<tr>
<td>\texttt{dmModeShowSecret}</td>
<td>Show records marked private.</td>
</tr>
</tbody>
</table>

Miscellaneous Constants

The following additional constants are used in conjunction with the Data Manager.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{dmDBNameLength}</td>
<td>32</td>
<td>Maximum length of a database name, including the null terminator. Database names must use only 7-bit ASCII characters (0x20 through 0x7E).</td>
</tr>
</tbody>
</table>
Data Manager Functions

DmArchiveRecord

**Purpose**
Mark a record as archived by leaving the record’s chunk intact and setting the delete bit for the next sync.

**Declared In**
DataMgr.h

**Prototype**
Err DmArchiveRecord (DmOpenRef dbP, UInt16 index)

**Parameters**
- `dbP` DmOpenRef to open database.
- `index` Which record to archive.

**Result**
Returns errNone if no error, or one of the following if an error occurs:

- `dmErrReadOnly`
- `dmErrIndexOutOfRange`
- `dmErrRecordArchived`
- `dmErrRecordDeleted`
- `memErrInvalidParam`

Some releases may display a fatal error message instead of returning the error code.

**Comments**
When a record is archived, the deleted bit is set but the chunk is not freed and the local ID is preserved. This way, the next time the user synchronizes with the desktop system, the desktop can save the record data on the PC before it permanently removes the record entry and data from the Palm Powered™ device.

Based on the assumption that a call to DmArchiveRecord indicates that you are finished with the record and aren’t going to refer to it again, this function sets the chunk’s lock count to zero.

**See Also**
DmRemoveRecord, DmDetachRecord, DmNewRecord, DmDeleteRecord
DmAttachRecord

Purpose  Attach an existing chunk ID handle to a database as a record.

Declared In  DataMgr.h

Prototype  Err DmAttachRecord (DmOpenRef dbP, UInt16 *atP, MemHandle newH, MemHandle *oldHP)

Parameters

- -> dbP  DmOpenRef to open database.
- <-> atP  Pointer to the index where the new record should be placed. Specify the value dmMaxRecordIndex to add the record to the end of the database.
- -> newH  Handle of the new record.
- <-> oldHP  If non-NULL upon entry, indicates that the record at *atP should be replaced. Upon return, contains the handle to the replaced record.

Result  Returns errNone if no error, or one of the following if an error occurs:

- dmErrMemError
- memErrChunkLocked
- memErrInvalidParam
- memErrNotEnoughSpace
- dmErrReadOnly
- dmErrNotRecordDB
- dmErrRecordInWrongCard
- dmErrIndexOutOfRange

Some releases may display a fatal error message instead of returning some of these error codes.

Comments  Given the handle of an existing chunk, this routine makes that chunk a new record in a database and sets the dirty bit. The
parameter atP points to an index variable. If oldHP is NULL, the new record is inserted at index *atP and all record indices that follow are shifted down. If *atP is greater than the number of records currently in the database, the new record is appended to the end and its index is returned in *atP. If oldHP is not NULL, the new record replaces an existing record at index *atP and the handle of the old record is returned in *oldHP so that the application can free it or attach it to another database.

This function is useful for cutting and pasting between databases.

See Also  
DmDetachRecord, DmNewRecord, DmNewHandle, DmFindSortPosition

**DmAttachResource**

**Purpose**  
Attach an existing chunk ID to a resource database as a new resource.

**Declared In**  
DataMgr.h

**Prototype**  
Err DmAttachResource (DmOpenRef dbP, MemHandle newH, DmResType resType, DmResID resID)

**Parameters**  
- > dbP  
  DmOpenRef to open database.
- > newH  
  Handle of new resource’s data.
- > resType  
  Type of the new resource.
- > resID  
  ID of the new resource.

**Result**  
Returns errNone if no error, or one of the following if an error occurs:

- dmErrMemError
- memErrChunkLocked
- memErrInvalidParam
- memErrNotEnoughSpace
- dmErrReadOnly
• dmErrRecordInWrongCard

Some releases may display a fatal error message instead of returning some of these error codes. All releases may display a fatal error message if the database is not a resource database.

Comments
Given the handle of an existing chunk with resource data in it, this routine makes that chunk a new resource in a resource database. The new resource will have the given type and ID.

See Also DmDetachResource, DmRemoveResource, DmNewHandle, DmNewResource

DmCloseDatabase

Purpose
Close a database.

Declared In
DataMgr.h

Prototype
Err DmCloseDatabase (DmOpenRef dbP)

Parameters
- dbP Database access pointer.

Result
Returns errNone if no error, or dmErrInvalidParam if an error occurs. Some releases may display a fatal error message instead of returning the error code.

Comments
This routine doesn’t unlock any records that were left locked. Records and resources should not be left locked. If a record/resource is left locked, you should not use its reference because the record can disappear during a HotSync operation or if the database is deleted by the user. To prevent the database from being deleted, you can use DmDatabaseProtect before closing.

If there is an overlay associated with the database passed in, DmCloseDatabase closes the overlay as well.

If the database has the recyclable bit set (dmHdrAttrRecyclable), DmCloseDatabase calls DmDeleteDatabase to delete it.
Compatibility

Starting with Palm OS 2.0, DmCloseDatabase updates the database’s modification date.

- On Palm OS 2.0, the modification date is updated if the database was opened with write access.
- On Palm OS 3.0 and higher, the modification date is updated only if a change has been made and the database was opened with write access. Changes that trigger an update include adding, deleting, archiving, rearranging, or resizing records, setting a record’s dirty bit in DmReleaseRecord, rearranging or deleting categories, or updating the database header fields using DmSetDatabaseInfo.

Under Palm OS 1.0, the modification date was never updated.

If you need to ensure that the modification date is updated the same way regardless of the operating system version, use DmSetDatabaseInfo to set the modification date explicitly.

See Also
DmOpenDatabase, DmDeleteDatabase, DmOpenDatabaseByTypeCreator

DmCreateDatabase

Purpose
Create a new database on the specified card with the given name, creator, and type.

Declared In
DataMgr.h

Prototype
Err DmCreateDatabase (UInt16 cardNo, const Char *nameP, UInt32 creator, UInt32 type, Boolean resDB)

Parameters
- `cardNo` The card number to create the database on.
- `nameP` Name of new database, up to 32 ASCII bytes long, including the null terminator (as specified by dmDBNameLength). Database names must use only 7-bit ASCII characters (0x20 through 0x7E).
- `creator` Creator of the database.
Data and Resource Manager
Data Manager Functions

- type Type of the database.
- resDB If true, create a resource database.

Result
Returns errNone if no error, or one of the following if an error occurs:

- dmErrInvalidDatabaseName
- dmErrAlreadyExists
- memErrCardNotPresent
- dmErrMemError
- memErrChunkLocked
- memErrInvalidParam
- memErrInvalidStoreHeader
- memErrNotEnoughSpace
- memErrRAMOnlyCard

May display a fatal error message if the master database list cannot be found.

Comments
Call this routine to create a new database on a specific card. If another database with the same name already exists in RAM store, this routine returns a dmErrAlreadyExists error code. Once created, the database ID can be retrieved by calling DmFindDatabase. The database can be opened using the database ID. To create a resource database instead of a record-based database, set the resDB Boolean to true.

After you create a database, it’s recommended that you call DmSetDatabaseInfo to set the version number. Databases default to version 0 if the version isn’t explicitly set.

See Also
DmCreateDatabaseFromImage, DmOpenDatabase, DmDeleteDatabase
**DmCreateDatabaseFromImage**

**Purpose**
Create an entire database from a single resource that contains an image of the database.

**Declared In**
DataMgr.h

**Prototype**
Err DmCreateDatabaseFromImage (MemPtr bufferP)

**Parameters**
- bufferP Pointer to locked resource containing database image.

**Result**
Returns errNone if no error.

**Comments**
An image is the same as a desktop file representation of a prc or pdb file.

This function is intended for applications in the ROM to install default databases after a hard reset. RAM-based applications that want to install a default database should install a pdb file separately to save storage heap space.

**See Also**
DmCreateDatabase, DmOpenDatabase

**DmDatabaseInfo**

**Purpose**
Retrieve information about a database.

**Declared In**
DataMgr.h

**Prototype**
Err DmDatabaseInfo (UInt16 cardNo, LocalID dbID, Char *nameP, UInt16 *attributesP, UInt16 *versionP, UInt32 *crDateP, UInt32 *modDateP, UInt32 *bckUpDateP, UInt32 *modNumP, LocalID *appInfoIDP, LocalID *sortInfoIDP, UInt32 *typeP, UInt32 *creatorP)

**Parameters**
- cardNo Number of the card the database resides on.
### Data and Resource Manager

#### Data Manager Functions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dbID</code></td>
<td>Database ID of the database.</td>
</tr>
<tr>
<td><code>&lt;nameP&gt;</code></td>
<td>The database’s name. Pass a pointer to a 32-byte character array for this parameter, or <code>NULL</code> if you don’t care about the name.</td>
</tr>
<tr>
<td><code>&lt;attributesP&gt;</code></td>
<td>The database’s attribute flags. The section “Database Attribute Constants” lists constants you can use to query the values returned in this parameter. Pass <code>NULL</code> for this parameter if you don’t want to retrieve it.</td>
</tr>
<tr>
<td><code>&lt;versionP&gt;</code></td>
<td>The application-specific version number. The default version number is 0. Pass <code>NULL</code> for this parameter if you don’t want to retrieve it.</td>
</tr>
<tr>
<td><code>&lt;crDateP&gt;</code></td>
<td>The date the database was created, expressed as the number of seconds since the first instant of Jan. 1, 1904. Pass <code>NULL</code> for this parameter if you don’t want to retrieve it.</td>
</tr>
<tr>
<td><code>&lt;modDateP&gt;</code></td>
<td>The date the database was last modified, expressed as the number of seconds since the first instant of Jan. 1, 1904. Pass <code>NULL</code> for this parameter if you don’t want to retrieve it.</td>
</tr>
<tr>
<td><code>&lt;bckUpDateP&gt;</code></td>
<td>The date the database was backed up, expressed as the number of seconds since the first instant of Jan. 1, 1904. Pass <code>NULL</code> for this parameter if you don’t want to retrieve it.</td>
</tr>
<tr>
<td><code>&lt;modNumP&gt;</code></td>
<td>The modification number, which is incremented every time a record in the database is added, modified, or deleted. Pass <code>NULL</code> for this parameter if you don’t want to retrieve it.</td>
</tr>
<tr>
<td><code>&lt;appInfoIDP&gt;</code></td>
<td>The local ID of the application info block, or <code>NULL</code>. The application info block is an optional field that the database may use to store application-specific information about the database. Pass <code>NULL</code> for this parameter if you don’t want to retrieve it.</td>
</tr>
</tbody>
</table>
sortInfoIDP  The local ID of the database’s sort table. This is an optional field in the database header. Pass NULL for this parameter if you don’t want to retrieve it.

typeP  The database’s type, specified when it is created. Pass NULL for this parameter if you don’t want to retrieve it.

creatorP  The database’s creator, specified when it is created. Pass NULL for this parameter if you don’t want to retrieve it.

Result  Returns errNone if no error, or dmErrInvalidParam if an error occurs.

Compatibility  Updating of the modification date differs based on which version of the OS your application is running on.

• Under Palm OS 1.0, the modification date is never updated.

• Under Palm OS 2.0, the modification date is updated every time a database opened with write access is closed.

• Beginning with Palm OS 3.0, the modification date is updated only if a change has been made to the database opened with write access. (The update still occurs upon closing the database.) Changes that trigger an update include adding, deleting, archiving, rearranging, or resizing records, setting a record’s dirty bit in DmReleaseRecord, rearranging or deleting categories, or updating the database header fields using DmSetDatabaseInfo.

If you need to ensure that the modification date is updated the same way regardless of the operating system version, use DmSetDatabaseInfo to set the modification date explicitly.

See Also  DmSetDatabaseInfo, DmDatabaseSize, DmOpenDatabaseInfo, DmFindDatabase, DmGetNextDatabaseByTypeCreator, TimSecondsToDateTime
DmDatabaseProtect

Purpose
Increment or decrement the database’s protection count.

Declared In
DataMgr.h

Prototype
Err DmDatabaseProtect (UInt16 cardNo, LocalID dbID, Boolean protect)

Parameters
- cardNo Card number of database to protect/unprotect.
- dbID Local ID of database to protect/unprotect.
- protect If true, protect count will be incremented. If false, protect count will be decremented.

Result
Returns errNone if no error, or one of the following if an error occurs:
- memErrCardNotPresent
- dmErrROMBased
- dmErrCantFind
- memErrNotEnoughSpace
- dmErrDatabaseNotProtected

Comments
This routine can be used to prevent a database from being deleted (by passing true for the protect parameter). It increments the protect count if protect is true and decrements it if protect is false. All true calls should be balanced by false calls before the application terminates.

Use this function if you want to keep a particular record or resource in a database locked down but don’t want to keep the database open. This information is kept in the dynamic heap, so all databases are “unprotected” at system reset.

If the database is a resource database that has an overlay associated with it for the current locale, the overlay is also protected or unprotected by this call.
Compatibility Implemented only if 2.0 New Feature Set is present. Overlay support is only available if 3.5 New Feature Set is present.

DmDatabaseSize

Purpose Retrieve size information on a database.

Declared In DataMgr.h

Prototype Err DmDatabaseSize (UInt16 cardNo, LocalID dbID, UInt32 *numRecordsP, UInt32 *totalBytesP, UInt32 *dataBytesP)

Parameters -> cardNo Card number the database resides on.
-> dbID Database ID of the database.
<- numRecordsP The total number of records in the database. Pass NULL for this parameter if you don’t want to retrieve it.
<- totalBytesP The total number of bytes used by the database including the overhead. Pass NULL for this parameter if you don’t want to retrieve it.
<- dataBytesP The total number of bytes used to store just each record’s data, not including overhead. Pass NULL for this parameter if you don’t want to retrieve it.

Result Returns errNone if no error, or dmErrMemError if an error occurs.

See Also DmDatabaseInfo, DmOpenDatabaseInfo, DmFindDatabase, DmGetNextDatabaseByTypeCreator
DmDeleteCategory

**Purpose**
Delete all records in a category. The category name is not changed.

**Declared In**
DataMgr.h

**Prototype**
Err DmDeleteCategory (DmOpenRef dbR, UInt16 categoryNum)

**Parameters**
- `dbR` Database access pointer.
- `categoryNum` Category of records to delete. Category masks such as `dmAllCategories` are invalid.

**Result**
Returns `errNone` if no error, or one of the following if an error occurs:
- `dmErrReadOnly`
- `memErrInvalidParam`

Some releases may display a fatal error message instead of returning the error code.

**Comments**
This function deletes all records in a category, but does not delete the category itself (note that it deletes the record data and header info, and doesn’t just set the deleted bit). For each record in the category, DmDeleteCategory marks the delete bit in the database header for the record and disposes of the record’s data chunk. The record entry in the database header remains, but its `localChunkID` is set to `NULL`.

If the category contains no records, this function does nothing and returns `errNone` to indicate success. The `categoryNum` parameter is assumed to represent a single category. If you pass a category mask to specify more than one category, this function interprets that value as a single category, finds no records to delete in that category, and returns `errNone`.

You can use the `DmRecordInfo` call to obtain a category index from a given record. For example:
DmOpenRef myDB;
UInt16 record, attr, category, total;

DmRecordInfo(myDB, record, &attr, NULL, NULL);
category = attr & dmRecAttrCategoryMask;
err = DmDeleteCategory(myDB, category);

Compatibility
Implemented only if 2.0 New Feature Set is present.

DmDeleteDatabase

Purpose
Delete a database and all its records.

Declared In
DataMgr.h

Prototype
Err DmDeleteDatabase (UInt16 cardNo, LocalID dbID)

Parameters
- -> cardNo Card number the database resides on.
- -> dbID Database ID.

Result
Returns errNone if no error, or one of the following if an error occurs:
- dmErrCantFind
- dmErrCantOpen
- memErrChunkLocked
- dmErrDatabaseOpen
- dmErrROMBased
- memErrInvalidParam
- memErrNotEnoughSpace

Comments
Call this routine to delete a database. This routine deletes the database, the application info block, the sort info block, and any other overhead information that is associated with this database. After deleting the database, this function enqueues a deferred sysNotifyDBDeletedEvent notification, which will be broadcast at the top of the event loop.
If the database has an overlay associated with it, this function does **not** delete the overlay. You can delete the overlay with a separate call to DmDeleteDatabase.

This routine accepts a database ID as a parameter. To determine the database ID, call either DmFindDatabase or DmGetDatabase with a database index.

**Compatibility**

The `sysNotifyDBDeletedEvent` notification is only broadcast if the **4.0 New Feature Set** is present.

**See Also**

DmDeleteRecord, DmRemoveRecord, DmRemoveResource, DmCreateDatabase, DmGetNextDatabaseByTypeCreator, DmFindDatabase

---

### DmDeleteRecord

**Purpose**

Delete a record’s chunk from a database but leave the record entry in the header and set the *delete* bit for the next sync.

**Declared In**

DataMgr.h

**Prototype**

```
Err DmDeleteRecord (DmOpenRef dbP, UInt16 index)
```

**Parameters**

- `-> dbP` DmOpenRef to open database.
- `-> index` Which record to delete.

**Result**

Returns `errNone` if no error, or one of the following if an error occurs:

- `dmErrReadOnly`
- `dmErrIndexOutOfRange`
- `dmErrRecord Archived`
- `dmErrRecordDeleted`
- `memErrInvalidParam`

Some releases may display a fatal error message instead of returning the error code.
**Comments**
Marks the delete bit in the database header for the record and disposes of the record’s data chunk. Does not remove the record entry from the database header, but simply sets the `localChunkID` of the record entry to `NULL`.

**See Also**
`DmDetachRecord`, `DmRemoveRecord`, `DmArchiveRecord`, `DmNewRecord`

---

**DmDetachRecord**

**Purpose**
Detach and orphan a record from a database but don’t delete the record’s chunk.

**Declared In**
`DataMgr.h`

**Prototype**
```
Err DmDetachRecord (DmOpenRef dbP, UInt16 index, MemHandle *oldHP)
```

**Parameters**
- `-> dbP` DmOpenRef to open.
- `-> index` Index of the record to detach.
- `<-> oldHP` Pointer to return handle of the detached record.

**Result**
Returns `errNone` if no error, or one of the following if an error occurs:
- `dmErrReadOnly`
- `dmErrIndexOutOfRange`
- `dmErrNotRecordDB`
- `memErrChunkLocked`
- `memErrInvalidParam`

Some releases may display a fatal error message instead of returning the error code.

**Comments**
This routine detaches a record from a database by removing its entry from the database header and returns the handle of the record’s data chunk in `oldHP`. Unlike `DmDeleteRecord`, this
routine removes its entry in the database header but it does not delete the actual record.

See Also  
DmAttachRecord, DmRemoveRecord, DmArchiveRecord, DmDeleteRecord

DmDetachResource

Purpose  
Detach a resource from a database and return the handle of the resource’s data.

Declared In  
DataMgr.h

Prototype  
Err DmDetachResource (DmOpenRef dbP, UInt16 index, MemHandle *oldHP)

Parameters
- -> dbP DmOpenRef to open database.
- -> index Index of resource to detach.
- <-> oldHP Pointer to return handle of the detached record.

Result  
Returns errNone if no error, or one of the following if an error occurs:

- dmErrReadOnly
- dmErrIndexOutOfRange
- dmErrCorruptDatabase
- memErrChunkLocked
- memErrInvalidParam

Some releases may display a fatal error message instead of returning the error code. All releases may display a fatal error message if the database is not a resource database.

Comments  
This routine detaches a resource from a database by removing its entry from the database header and returns the handle of the resource’s data chunk in *oldHP.

See Also  
DmAttachResource, DmRemoveResource
Data and Resource Manager

Data Manager Functions

DmFindDatabase

Purpose
Return the database ID of a database by card number and name.

Declared In
DataMgr.h

Prototype
LocalID DmFindDatabase (UInt16 cardNo,
const Char *nameP)

Parameters
- cardNo
  Number of card to search.
- nameP
  Name of the database to look for.

Result
Returns the database ID. If the database can’t be found, this function
returns 0, and DmGetLastError returns an error code indicating the
reason for failure.

See Also
DmGetNextDatabaseByTypeCreator, DmDatabaseInfo,
DmOpenDatabase

DmFindRecordByID

Purpose
Return the index of the record with the given unique ID.

Declared In
DataMgr.h

Prototype
Err DmFindRecordByID (DmOpenRef dbP,
UInt32 uniqueID, UInt16 *indexP)

Parameters
- dbP
  Database access pointer.
- uniqueID
  Unique ID to search for.
- indexP
  Return index.

Result
Returns 0 if found, otherwise dmErrUniqueIDNotFound. May
display a fatal error message if the unique ID is invalid.

See Also
DmQueryRecord, DmGetRecord, DmRecordInfo
DmFindResource

Purpose
Search the given database for a resource by type and ID, or by pointer if it is non-NULL.

Declared In
DataMgr.h

Prototype
UInt16 DmFindResource (DmOpenRef dbP, DmResType resType, DmResID resID, MemHandle resH)

Parameters
-> dbP	Open resource database access pointer.
-> resType	Type of resource to search for.
-> resID	ID of resource to search for.
-> resH	Pointer to locked resource, or NULL.

Result
Returns index of resource in resource database, or 0xFFFF if not found.
May display a fatal error message if the database is not a resource database.

Comments
Use this routine to find a resource in a particular resource database by type and ID or by pointer. It is particularly useful when you want to search only one database for a resource and that database is not the topmost one.

IMPORTANT: This function searches for the resource only in the database you specify. If you pass a pointer to a base resource database, its overlay is not searched. To search both a base database and its overlay for a localized resource, use DmGet1Resource instead of this function.

If resH is NULL, the resource is searched for by type and ID.
If resH is not NULL, resType and resID are ignored and the index of the given locked resource is returned.
Once the index of a resource is determined, it can be locked down and accessed by calling `DmGetResourceIndex`.

See Also  
`DmGetResource`, `DmSearchResource`, `DmResourceInfo`, `DmGetResourceIndex`, `DmFindResourceType`

### DmFindResourceType

**Purpose**  
Search the given database for a resource by type and type index.

**Declared In**  
DataMgr.h

**Prototype**  
```c
UInt16 DmFindResourceType (DmOpenRef dbP, DmResType resType, UInt16 typeIndex)
```

**Parameters**  
- `dbP`  
  Open resource database access pointer.
- `resType`  
  Type of resource to search for.
- `typeIndex`  
  Index of given resource type.

**Result**  
Index of resource in resource database, or 0xFFFF if not found.

May display a fatal error message if the database is not a resource database.

**Comments**  
Use this routine to retrieve all the resources of a given type in a resource database. By starting at `typeIndex` 0 and incrementing until an error is returned, the total number of resources of a given type and the index of each of these resources can be determined. Once the index of a resource is determined, it can be locked down and accessed by calling `DmGetResourceIndex`.
**Data and Resource Manager**

*Data Manager Functions*

**IMPORTANT:** This function searches for resources only in the database you specify. If you pass a pointer to a base resource database, its overlay is **not** searched. To search both a base database and its overlay for a localized resource, use `DmGet1Resource` instead of this function.

---

**See Also**

`DmGetResource`, `DmSearchResource`, `DmResourceInfo`, `DmGetResourceIndex`, `DmFindResource`

---

**DmFindSortPosition**

**Purpose**

Returns where in a sorted list of records a given record would be located. Useful to find where to insert a record with `DmAttachRecord`. Uses a binary search.

**Declared In**

`DataMgr.h`

**Prototype**

```c
UInt16 DmFindSortPosition (DmOpenRef dbP, void *newRecord, SortRecordInfoPtr newRecordInfo, DmComparF *compar, Int16 other)
```

**Parameters**

- `-> dB P` Database access pointer.
- `-> newRecord` Pointer to the new record.
- `-> newRecordInfo` Sort information about the new record. See `SortRecordInfoType`.
- `-> compar` Pointer to comparison function. See `DmComparF`.
Any value the application wants to pass to the comparison function. This parameter is often used to indicate a sort direction (ascending or descending).

**Result**

The position where the record should be inserted.

The position should be viewed as between the record returned and the record before it. Note that the return value may be one greater than the number of records.

**Comments**

If `newRecord` has the same key as another record in the database, `DmFindSortPosition` assumes that `newRecord` should be inserted after that record. If there are several records with the same key, `newRecord` is inserted after all of them. For this reason, if you use `DmFindSortPosition` to search for the location of a record that you know is already in the database, you must subtract 1 from the result. (Be sure to check that the value is not 0.)

If there are deleted records in the database, `DmFindSortPosition` only works if those records are at the end of the database. `DmFindSortPosition` always assumes that a deleted record is greater than or equal to any other record.

**Compatibility**

Implemented only if [2.0 New Feature Set](#) is present.

**See Also**

[DmFindSortPositionV10](#)

---

**DmFindSortPositionV10**

**Purpose**

Return where a record should be. Useful to find where to insert a record with `DmAttachRecord`. Uses a binary search.

**Declared In**

`DataMgr.h`

**Prototype**

```c
UInt16 DmFindSortPositionV10 (DmOpenRef dbP, void *newRecord, DmComparF *compar, Int16 other)
```

**Parameters**

- `-> dbP` Database access pointer.
Data and Resource Manager
Data Manager Functions

-> newRecord  Pointer to the new record.
-> compar  Pointer to comparison function. See DmComparF.
-> other  Any value the application wants to pass to the comparison function.

Result  Returns the position where the record should be inserted. The position should be viewed as between the record returned and the record before it. Note that the return value may be one greater than the number of records.

Comments  If there are deleted records in the database, DmFindSortPositionV10 only works if those records are at the end of the database. DmFindSortPositionV10 always assumes that a deleted record is greater than or equal to any other record.

Compatibility  This function corresponds to the 1.0 version of DmFindSortPosition.

See Also  DmFindSortPosition, DmQuickSort, DmInsertionSort

DmGetAppInfoID

Purpose  Return the local ID of the application info block.

Declared In  DataMgr.h

Prototype  LocalID DmGetAppInfoID (DmOpenRef dbP).

Parameters  -> dbP  Database access pointer.

Result  Returns local ID of the application info block. The application info block is an optional field that the database may use to store application-specific information about the database; if the database doesn’t have an application info block, DmGetAppInfoID returns zero.

See Also  DmDatabaseInfo, DmOpenDatabase
**Data and Resource Manager**

*Data Manager Functions*

---

### DmGetDatabase

**Purpose**

Return the database header ID of a database by index and card number.

**Declared In**

DataMgr.h

**Prototype**

```
LocalID DmGetDatabase (UInt16 cardNo,
                       UInt16 index)
```

**Parameters**

- `-> cardNo` Card number of database.
- `-> index` Index of database.

**Result**

Returns the database ID, or 0 if an invalid parameter is passed.

**Comments**

Call this routine to retrieve the database ID of a database by index. The index should range from 0 to `DmNumDatabases`-1. This routine is useful for getting a directory of all databases on a card. The databases returned may reside in either the ROM or the RAM. The order in which databases are returned is not fixed; therefore, you should not rely on receiving a list of databases in a particular order.

**See Also**

DmOpenDatabase, DmNumDatabases, DmDatabaseInfo, DmDatabaseSize

---

### DmGetDatabaseLockState

**Purpose**

Return information about the number of locked and busy records in a database.

**Declared In**

DataMgr.h

**Prototype**

```
void DmGetDatabaseLockState (DmOpenRef dbR,
                             UInt8 *highest, UInt32 *count, UInt32 *busy)
```

**Parameters**

- `-> dbR` Database access pointer.
**Data and Resource Manager**

**Data Manager Functions**

- **highest**
  The highest lock count found for all of the records in the database. If a database has two records, one has a lock count of 2 and one has a lock count of 1, the highest lock count is 2. Pass NULL for this parameter if you don’t want to retrieve it.

- **count**
  The number of records that have the lock count that is returned in the highest parameter. Pass NULL for this parameter if you don’t want to retrieve it.

- **busy**
  The number of records that have the busy bit set. Pass NULL for this parameter if you don’t want to retrieve it.

**Result**
No return value. Returns all information in the parameters you pass.

**Comments**
This function is intended to be used for debugging purposes. You can use it to obtain information about how many records are busy and how much locking occurs.

**Compatibility**
Implemented only if 3.2 New Feature Set is present.

---

**DmGetLastError**

**Purpose**
Return error code from last data manager call.

**Declared In**
DataMgr.h

**Prototype**
Err DmGetLastError (void)

**Parameters**
None.

**Result**
Error code from last unsuccessful data manager call.

**Comments**
Use this routine to determine why a data manager call failed. In particular, calls like DmGetRecord return 0 if unsuccessful, so
calling `DmGetLastErr` is the only way to determine why they failed.

Note that `DmGetLastErr` does not always reflect the error status of the last data manager call. Rather, it reflects the error status of data manager calls that don’t return an error code. For some of those calls, the saved error code value is not set to 0 when the call is successful.

For example, if a call to `DmOpenDatabaseByTypeCreator` returns NULL for database reference (that is, it fails), `DmGetLastErr` returns something meaningful; otherwise, it returns the error value of some previous data manager call.

Only the following data manager functions currently affect the value returned by `DmGetLastErr`:

- `DmFindDatabase`
- `DmOpenDatabase`
- `DmQueryRecord`
- `DmQueryNextInCategory`
- `DmSeekRecordInCategory`
- `DmGetResource`
- `DmNewResource`
- `DmNewHandle`
- `DmResizeResource`
- `DmOpenDatabaseByTypeCreator`
- `DmNewRecord`
- `DmGetRecord`
- `DmPositionInCategory`
- `DmResizeRecord`
- `DmGet1Resource`
- `DmGetResourceIndex`
- `DmOpenDBNoOverlay`
DmGetNextDatabaseByTypeCreator

Purpose
Return a database header ID and card number given the type and/or creator. This routine searches all memory cards for a match.

Declared In
DataMgr.h

Prototype
Err DmGetNextDatabaseByTypeCreator
(Boolean newSearch, DmSearchStatePtr stateInfoP,
UInt32 type, UInt32 creator,
Boolean onlyLatestVers, UInt16 *cardNoP,
LocalID *dbIDP)

Parameters
-> newSearch true if starting a new search.
<-> stateInfoP If newSearch is false, this must point to the same data used for the previous invocation.
-> type Type of database to search for, pass 0 as a wildcard.
-> creator Creator of database to search for, pass 0 as a wildcard.
-> onlyLatestVers If true, only the latest version of a database with a given type and creator is returned.
<- cardNoP On exit, the card number of the found database.
<- dbIDP Database local ID of the found database.

Result
Returns errNone if no error, or dmErrCantFind if no matches were found.

Comments
You may need to call this function successively to discover all databases having a specified type/creator pair.
To start the search, pass true for newSearch. Allocate a DmSearchStateType structure and pass it as the stateInfoP parameter. DmGetNextDatabaseByTypeCreator stores private information in stateInfoP and uses it if the search is continued.
To continue a search where the previous one left off, pass false for newSearch and pass the same stateInfoP that you used during the previous call to this function.

You can pass NULL as a wildcard operator for the type or creator parameter to conduct searches of wider scope. If the type parameter is NULL, this routine can be called successively to return all databases of the given creator. If the creator parameter is NULL, this routine can be called successively to return all databases of the given type. You can also pass NULL as the value for both of these parameters to return all available databases without regard to type or creator.

Because databases are scattered freely throughout memory space, they are not returned in any particular order—any database matching the specified type/creator criteria can be returned. Thus, if the value of the onlyLatestVers parameter is false, this function may return a database which is not the most recent version matching the specified type/creator pair. To obtain only the latest version of a database matching the search criteria, set the value of the onlyLatestVers parameter to true.

When determining which is the latest version of the database, RAM databases are considered newer than ROM databases that have the same version number. Because of this, you can replace any ROM-based application with your own version of it. Also, a RAM database on card 1 is considered newer than a RAM database on card 0 if the version numbers are identical.

**WARNING!** Don’t create or delete a database while using DmGetNextDatabaseByTypeCreator to iterate through the existing databases. This could cause databases to be skipped, or it could result in a given database being returned more than once.

**Compatibility**

In Palm OS version 3.1 and higher, if onlyLatestVers is true, you only receive one matching database for each type/creator pair. In version 3.0 and earlier, you could receive multiple matching databases if onlyLatestVers was true.

Note that the behavior is different only when you have specified a value for both type and creator and onlyLatestVers is true.
For example, suppose your application maintains three databases that all have the same type, creator, and version number and you write this code to process them in some way:

```c
DmSearchStateType state;
Boolean latestVer;
UInt16 card;
LocalID currentDB = 0;

theErr = DmGetNextDatabaseByTypeCreator(true,
   &state, myType, myCreator, latestVer, &card,
   &currentDB);
while (!theErr && currentDB) {
    /* do something with currentDB */
    /* now get the next DB */
    theErr = DmGetNextDatabaseByTypeCreator(
        false, &state, myType, myCreator,
        vlatestVer, &card, &currentDB);
}
```

If `latestVer` is `false`, then your code will work the same on all versions of Palm OS and will return all three databases whose type and creator match those specified. If `latestVer` is `true`, this code returns all three databases on Palm OS version 3.0 and earlier, but only returns one database on version 3.1 and higher. (Exactly which database it returns is unspecified.)

If you expect multiple databases to match your search criteria, make sure you call `DmGetNextDatabaseByTypeCreator` in one of the following ways to ensure that your code operates the same on all Palm OS versions:

- Set `onlyLatestVers` to `false` if you specify both a type and creator.
- Specify `NULL` for either the `type` or `creator` parameter (or both).

**See Also**  
[DmGetDatabase](#), [DmFindDatabase](#), [DmDatabaseInfo](#),  
[DmOpenDatabaseByTypeCreator](#), [DmDatabaseSize](#)
DmGetRecord

Purpose  Return a handle to a record by index and mark the record busy.

Declared In  DataMgr.h

Prototype  MemHandle DmGetRecord (DmOpenRef dbP, Uint16 index)

Parameters  
- dbP  DmOpenRef to open database.
- index  Which record to retrieve.

Result  Returns a handle to record data. If another call to DmGetRecord for the same record is attempted before the record is released, NULL is returned and DmGetLastError returns an error code indicating the reason for failure.

Comments  Returns a handle to given record and sets the busy bit for the record.

If the record is ROM-based (pointer accessed), this routine makes a fake handle to it and stores this handle in the DmAccessType structure.

DmReleaseRecord should be called as soon as the caller finishes viewing or editing the record.

See Also  DmSearchRecord, DmFindRecordByID, DmRecordInfo, DmReleaseRecord, DmQueryRecord
**DmGetResource**

**Purpose**
Search all open resource databases and return a handle to a resource, given the resource type and ID.

**Declared In**
DataMgr.h

**Prototype**
MemHandle DmGetResource (DmResType type, DmResID resID)

**Parameters**
- **type**  
The resource type.
- **resID**  
The resource ID.

**Result**
Handle to resource data. If the specified resource cannot be found, this function returns NULL and *DmGetLastError* returns an error code indicating the reason for failure.

**Comments**
Searches all open resource databases starting with the most recently opened one for a resource of the given type and ID. If found, the resource handle is returned. The application should call *DmReleaseResource* as soon as it finishes accessing the resource data. The resource handle is not locked by this function.

This function always returns the resource located in the overlay if any open overlay has a resource matching that type and ID. If there is no overlay version of the resource, this function returns the resource from the base database.

**See Also**  
*DmGet1Resource*, *DmReleaseResource*, *ResLoadConstant*
DmGetResourceIndex

Purpose
Return a handle to a resource by index.

Declared In
DataMgr.h

Prototype
MemHandle DmGetResourceIndex (DmOpenRef dbP, UInt16 index)

Parameters
-> dbP  
Access pointer to open database.

-> index  
Index of the resource whose handle you want.

Result
Handle to resource data. If the specified index is out of range, this function returns NULL and DmGetLastErr returns an error code indicating the reason for failure.

May display a fatal error message if the database is not a resource database.

IMPORTANT: This function accesses the resource only in the database you specify. If you pass a pointer to a base resource database, its overlay is not accessed. Therefore, you should use care when using this function to access a potentially localized resource. You can use DmSearchResource to obtain a pointer to the overlay database if the resource is localized; however, it’s more convenient to use DmGetResource or DmGet1Resource.

See Also
DmFindResource, DmFindResourceType, DmSearchResource
**DmGet1Resource**

**Purpose**
Search the most recently opened resource database and return a handle to a resource given the resource type and ID.

**Declared In**
DataMgr.h

**Prototype**
MemHandle DmGet1Resource (DmResType type, DmResID resID)

**Parameters**
- type The resource type.
- resID The resource ID.

**Result**
Handle to resource data. If unsuccessful, this function returns NULL and DmGetLastError returns an error code indicating the reason for failure.

**Comments**
Searches the most recently opened resource database for a resource of the given type and ID. If the database has an overlay associated with it, the overlay is searched first, and then the base database is searched if the overlay does not contain the resource. If found, the resource handle is returned. The application should call DmReleaseResource as soon as it finishes accessing the resource data. The resource handle is not locked by this function.

**See Also**
DmGetResource, DmReleaseResource, ResLoadConstant

**DmInsertionSort**

**Purpose**
Sort records in a database.

**Declared In**
DataMgr.h

**Prototype**
Err DmInsertionSort (DmOpenRef dbR, DmComparF *compar, Int16 other)

**Parameters**
- dbR Database access pointer.
Data and Resource Manager
Data Manager Functions

-> compar

Comparison function. See DmCompareF.

-> other

Any value the application wants to pass to the comparison function. This parameter is often used to indicate a sort direction (ascending or descending).

Result

Returns errNone if no error, or one of the following if an error occurs:

- dmErrReadOnly
- dmErrNotRecordDB

Some releases may display a fatal error message instead of returning the error code.

Comments

Deleted records are placed last in any order. All others are sorted according to the passed comparison function. Only records which are out of order move. Moved records are moved to the end of the range of equal records. If a large number of records are being sorted, try to use the quick sort.

The following insertion-sort algorithm is used: Starting with the second record, each record is compared to the preceding record. Each record not greater than the last is inserted into sorted position within those already sorted. A binary insertion is performed. A moved record is inserted after any other equal records.

See Also

DmQuickSort

DmMoveCategory

Purpose

Move all records in a category to another category.

Declared In

DataMgr.h

Prototype

Err DmMoveCategory (DmOpenRef dbP, UInt16 toCategory, UInt16 fromCategory, Boolean dirty)

Parameters

-> dbP

DmOpenRef to open database.
Data and Resource Manager
Data Manager Functions

-DtoCategory Category to which the records should be added.
-> fromCategory Category from which to remove records.
-> dirty If true, set the dirty bit.

Result Returns 0 if successful, or dmErrReadOnly if the database is in read-only mode. Some releases may display a fatal error message instead of returning the error code.

Comments If dirty is true, the moved records are marked as dirty.

The toCategory and fromCategory parameters hold category index values. You can learn which category a record is in with the DmRecordInfo call and use that value in this function. For example, the following code, ensures that the records rec1 and rec2 are in the same category:

DmOpenRef myDB;
UInt16 rec1, rec2;
UInt16 rec1Attr, rec2Attr;
UInt16 category1, category2;

DmRecordInfo (myDB, rec1, &rec1Attr, NULL, NULL);
category1 = rec1Attr & dmRecAttrCategoryMask;
DmRecordInfo(myDB, rec2, &rec2Attr, NULL, NULL);
category2 = rec2Attr & dmRecAttrCategoryMask;
if (category1 != category2)
    DmMoveCategory(myDB, category1, category2, true);
DmMoveRecord

**Purpose**
Move a record from one index to another.

**Declared In**
DataMgr.h

**Prototype**
Err DmMoveRecord (DmOpenRef dbP, UInt16 from, UInt16 to)

**Parameters**
- `dbP` DmOpenRef to open database.
- `from` Index of record to move.
- `to` Where to move the record.

**Result**
Returns `errNone` if no error, or one of the following if an error occurs:
- `dmErrReadOnly`
- `dmErrIndexOutOfRange`
- `dmErrNotRecordDB`
- `dmErrMemError`
- `memErrInvalidParam`
- `memErrChunkLocked`

Some releases may display a fatal error message instead of returning the error code.

**Comments**
Insert the record at the `to` index and move other records down. The `to` position should be viewed as an insertion position. This value may be one greater than the index of the last record in the database. In cases where `to` is greater than `from`, the new index of the record becomes `to-1` after the move is complete.
**DmNewHandle**

**Purpose**
Attempt to allocate a new chunk in the same data heap or card as the database header of the passed database access pointer. If there is not enough space in that data heap, try other heaps.

**Declared In**
DataMgr.h

**Prototype**
MemHandle DmNewHandle (DmOpenRef dbP, UInt32 size)

**Parameters**
- `dbP` DmOpenRef to open database.
- `size` Size of new handle.

**Result**
Returns the chunkID of new chunk. If an error occurs, returns 0, and `DmGetLastErr` returns an error code indicating the reason for failure.

**Comments**
Allocates a new handle of the given size. Ensures that the new handle is in the same memory card as the given database. This guarantees that you can attach the handle to the database as a record to obtain and save its LocalID in the appInfoID or sortInfoID fields of the header.

The handle should be attached to a database as soon as possible. If it is not attached to a database and the application crashes, the memory used by the new handle is unavailable until the next soft reset.
DmNewRecord

**Purpose**
Return a handle to a new record in the database and mark the record busy.

**Declared In**
DataMgr.h

**Prototype**
MemHandle DmNewRecord (DmOpenRef dbP, UInt16 *atP, UInt32 size)

**Parameters**
- `dbP` DmOpenRef to open database.
- `atP` Pointer to index where new record should be placed. Specify the value dmMaxRecordIndex to add the record to the end of the database.
- `size` Size of new record.

**Result**
Handle to record data. If an error occurs, this function returns 0 and DmGetLastError returns an error code indicating the reason for failure.

Some releases may display a fatal error message if the database is opened in read-only mode or it is a resource database.

**Comments**
Allocates a new record of the given size, and returns a handle to the record data. The parameter `atP` points to an index variable. The new record is inserted at index `*atP` and all record indices that follow are shifted down. If `*atP` is greater than the number of records currently in the database, the new record is appended to the end and its index is returned in `*atP`.

Both the busy and dirty bits are set for the new record and a unique ID is automatically created.

DmReleaseRecord should be called as soon as the caller finishes viewing or editing the record.

**See Also**
DmAttachRecord, DmRemoveRecord, DmDeleteRecord
DmNewResource

**Purpose**  Allocate and add a new resource to a resource database.

**Declared In**  DataMgr.h

**Prototype**  MemHandle DmNewResource (DmOpenRef dbP, DmResType resType, DmResID resID, UInt32 size)

**Parameters**
- `-> dbP`  DmOpenRef to open database.
- `-> resType`  Type of the new resource.
- `-> resID`  ID of the new resource.
- `-> size`  Desired size of the new resource.

**Result**  Returns a handle to the new resource. If an error occurs, this function returns NULL and DmGetLastErr returns an error code indicating the reason for failure.

May display a fatal error message if the database is not a resource database.

**Comments**  Allocates a memory chunk for a new resource and adds it to the given resource database. The new resource has the given type and ID. If successful, the application should call DmReleaseResource as soon as it finishes initializing the resource.

**See Also**  DmAttachResource, DmRemoveResource
**DmNextOpenDatabase**

**Purpose**
Return DmOpenRef to next open database for the current task.

**Declared In**
DataMgr.h

**Prototype**
DmOpenRef DmNextOpenDatabase (DmOpenRef currentP)

**Parameters**
- `currentP` Current database access pointer or NULL.

**Result**
DmOpenRef to next open database, or NULL if there are no more.

**Comments**
Call this routine successively to get the DmOpenRefs of all open databases. Pass NULL for currentP to get the first one. Applications don’t usually call this function, but is useful for system information.

**See Also**
DmOpenDatabaseInfo, DmDatabaseInfo

**DmNextOpenResDatabase**

**Purpose**
Return access pointer to next open resource database in the search chain.

**Declared In**
DataMgr.h

**Prototype**
DmOpenRef DmNextOpenResDatabase (DmOpenRef dbP)

**Parameters**
- `dbP` Database reference, or 0 to start search from the top.

**Result**
Pointer to next open resource database.

**Comments**
Returns pointer to next open resource database. To get a pointer to the first one in the search chain, pass NULL for `dbP`. This is the database that is searched when DmGet1Resource is called.
If you use this function to access a resource database that might have an overlay associated with it, be careful how you use the result. The DmOpenRef returned by this function is a pointer to the overlay database, not the base database. If you subsequently pass this pointer to DmFindResource, you’ll receive a handle to the overlaid resource. If you’re searching for a resource that is found only in the base, you won’t find it. Instead, always use DmGetResource or DmGet1Resource to obtain a resource. Both of those functions search both the overlay databases and their associated base databases.

**DmNumDatabases**

**Purpose** Determine how many databases reside on a memory card.

**Declared In** DataMgr.h

**Prototype**

```c
UInt16 DmNumDatabases (UInt16 cardNo)
```

**Parameters**

- `-> cardNo` Number of the card to check.

**Result** The number of databases found.

**Comments** This routine is helpful for getting a directory of all databases on a card. The routine DmGetDatabase accepts an index from 0 to DmNumDatabases -1 and returns a database ID by index.

**See Also** DmGetDatabase
**DmNumRecords**

**Purpose**
Return the number of records in a database.

**Declared In**
DataMgr.h

**Prototype**
```c
UInt16 DmNumRecords (DmOpenRef dbP)
```

**Parameters**
- `-> dbP` DmOpenRef to open database.

**Result**
The number of records in a database.

**Comments**
Records that have the deleted bit set (that is, records that will be deleted during the next synchronization because the user has marked them deleted) are included in the count. If you want to exclude these records from your count, use `DmNumRecordsInCategory` and pass `dmAllCategories` as the category.

**See Also**
`DmNumRecordsInCategory`, `DmRecordInfo`, `DmSetRecordInfo`

---

**DmNumRecordsInCategory**

**Purpose**
Return the number of records of a specified category in a database.

**Declared In**
DataMgr.h

**Prototype**
```c
UInt16 DmNumRecordsInCategory (DmOpenRef dbP, UInt16 category)
```

**Parameters**
- `-> dbP` DmOpenRef to open database.
- `-> category` Category index.

**Result**
The number of records in the category.
**Data and Resource Manager**

**Data Manager Functions**

**Comments**

Because this function must examine all records in the database, it can be slow to return, especially when called on a large database. Records that have the deleted bit set are not counted, and if the user has specified to hide or mask private records, private records are not counted either.

You can use the `DmRecordInfo` call to obtain a category index from a given record. For example:

```c
DmOpenRef myDB;
UInt16 record, attr, category, total;

DmRecordInfo(myDB, record, &attr, NULL, NULL);
category = attr & dmRecAttrCategoryMask;
total = DmNumRecordsInCategory(myDB, category);
```

**See Also**

`DmNumRecords`, `DmQueryNextInCategory`, `DmPositionInCategory`, `DmSeekRecordInCategory`, `DmMoveCategory`

---

**DmNumResources**

**Purpose**

Return the total number of resources in a given resource database.

**Declared In**

DataMgr.h

**Prototype**

```c
UInt16 DmNumResources (DmOpenRef dbP)
```

**Parameters**

- `-> dbP` DmOpenRef to open database.

**Result**

The total number of resources in the given database.

May display a fatal error message if the database is not a resource database.

**Comments**

`DmNumResources` only counts the resources in the database indicated by the `DmOpenRef` parameter. If the database is a resource database that has an overlay associated with it, this function only returns the number of resources in the base database, not in the overlay.
DmOpenDatabase

**Purpose**
Open a database and return a reference to it. If the database is a resource database, also open its overlay for the current locale.

**Declared In**
DataMgr.h

**Prototype**
DmOpenRef DmOpenDatabase (UInt16 cardNo, LocalID dbID, UInt16 mode)

**Parameters**
- **cardNo** Card number database resides on.
- **dbID** The database ID of the database.
- **mode** Which mode to open database in (see “Open Mode Constants”).

**Result**
Returns DmOpenRef to open database. May display a fatal error message if the database parameter is NULL. On all other errors, this function returns 0 and DmGetLastErr returns an error code indicating the reason for failure.

**Comments**
Call this routine to open a database for reading or writing.

This routine returns a DmOpenRef which must be used to access particular records in a database. If unsuccessful, 0 is returned and the cause of the error can be determined by calling DmGetLastErr.

When you use this routine to open a resource database in read-only mode, it also opens the overlay associated with this database for the current locale, if it exists. (The function OmGetCurrentLocale returns the current locale.) Overlays are resource databases typically used to localize applications, shared libraries, and panels. They have the same creator as the base database, a type of 'ovly' (symbolically named omOverlayDBType), and contain resources with the same IDs and types as the resources in the base database.

When you request a resource from the database using DmGetResource or DmGet1Resource, the overlay is searched first. If the overlay contains a resource for the given ID, it is returned. If not, the resource from the base database is returned.
The `DmOpenRef` returned by this function is the pointer to the base database, not to the overlay database, so care should be taken when passing this pointer to functions such as `DmFindResource` because this circumvents the overlay.

It’s possible to create a “stripped” base resource database, one that does not contain any user interface resources. `DmOpenDatabase` only opens a stripped database if its corresponding overlay exists. If the overlay does not exist or if the overlay doesn’t match the resource database, `DmOpenDatabase` returns `NULL` and `DmGetLastError` returns the error code `omErrBaseRequiresOverlay`.

If you open a resource database in a writable mode, the associated overlay is not opened. If you make changes to the resource database, the overlay database is invalidated if those changes affect any resources that are also in the overlay. This means that on future occasions where you open the resource database in read-only mode, the overlay will not be opened because Palm OS considers it to be invalid.

**TIP:** If you want to prevent your resource database from being overlaid, include an 'xprf' resource (symbolically named `sysResTExtPrefs`) in the database with the ID 0 (`sysResIDExtPrefs`) and set its disableOverlays flag. This resource is defined in `UIResources.r`.

**Compatibility**

Overlay support is only available if 3.5 New Feature Set is present. On earlier releases, this function opens resource databases without looking for an associated overlay.

If 4.0 New Feature Set is present, when `DmOpenDatabase` attempts to open a stripped resource database and cannot find an overlay for it, it searches for an overlay matching the default locale if the system locale is different from the default locale.

**See Also**

`DmCloseDatabase`, `DmCreateDatabase`, `DmFindDatabase`, `DmOpenDatabaseByTypeCreator`, `DmDeleteDatabase`, `DmOpenDBNoOverlay`
DmOpenDatabaseByTypeCreator

**Purpose**
Open the most recent revision of a database with the given type and creator. If the database is a resource database, also open its overlay for the current locale.

**Declared In**
DataMgr.h

**Prototype**
DmOpenRef DmOpenDatabaseByTypeCreator
(UInt32 type, UInt32 creator, UInt16 mode)

**Parameters**
- **type**
  Type of database.
- **creator**
  Creator of database.
- **mode**
  Which mode to open database in (see “Open Mode Constants”).

**Result**
DmOpenRef to open database. If the database couldn’t be found, this function returns 0 and DmGetLastError returns an error code indicating the reason for failure.

**Comments**
If you use this routine to open a resource database in read-only mode, it also opens the overlay associated with this database for the current locale. See DmOpenDatabase for more information on overlays and resource databases.

**Compatibility**
Overlay support is only available if 3.5 New Feature Set is present. On earlier releases, this function opens resource databases without looking for an associated overlay.

**See Also**
DmCreateDatabase, DmOpenDatabase, DmOpenDatabaseInfo, DmCloseDatabase, DmOpenDBNoOverlay
Data and Resource Manager

Data Manager Functions

DmOpenDatabaseInfo

**Purpose**
Retrieve information about an open database.

**Declared In**
DataMgr.h

**Prototype**
```
Err DmOpenDatabaseInfo (DmOpenRef dbP,
LocalID *dbIDP, UInt16 *openCountP, UInt16 *modeP,
UInt16 *cardNoP, Boolean *resDBP)
```

**Parameters**
- `dbP` DmOpenRef to open database.
- `dbIDP` The ID of the database. Pass NULL for this parameter if you don’t want to retrieve this information.
- `openCountP` The number of applications that have this database open. Pass NULL for this parameter if you don’t want to retrieve this information.
- `modeP` The mode used to open the database (see “Open Mode Constants”). Pass NULL for this parameter if you don’t want to retrieve this information.
- `cardNoP` The number of the card on which this database resides. Pass NULL for this parameter if you don’t want to retrieve this information.
- `resDBP` If true upon return, the database is a resource database, false otherwise. Pass NULL for this parameter if you don’t want to retrieve this information.

**Result**
Returns errNone if no error.

**See Also**
DmDatabaseInfo
**DmOpenDBNoOverlay**

**Purpose**  Open a database and return a reference to it.

**Declared In**  DataMgr.h

**Prototype**  

```c
DmOpenRef DmOpenDBNoOverlay (UInt16 cardNo, 
LocalID dbID, UInt16 mode)
```

**Parameters**

- `cardNo`  Card number database resides on.
- `dbID`  The database ID of the database.
- `mode`  Which mode to open database in (see “Open Mode Constants”).

**Result**  DmOpenRef to open database. May display a fatal error message if the database parameter is NULL. On all other errors, this function returns 0 and DmGetLastErr returns an error code indicating the reason for failure.

**Comments**  Call this routine to open a database for reading or writing, while ignoring any overlay databases that might be associated with it. This routine returns a DmOpenRef which must be used to access particular records in a database. If unsuccessful, 0 is returned and the cause of the error can be determined by calling DmGetLastErr.

**Compatibility**  Implemented only if 3.5 New Feature Set is present.

**See Also**  
DmCloseDatabase, DmCreateDatabase, DmFindDatabase, DmOpenDatabaseByTypeCreator, DmDeleteDatabase, DmOpenDatabase
DmPositionInCategory

Purpose
Return a position of a record within the specified category.

Declared In
DataMgr.h

Prototype
UInt16 DmPositionInCategory (DmOpenRef dbP, Uint16 index, Uint16 category)

Parameters
- dbP DmOpenRef to open database.
- index Index of the record.
- category Index of category to search.

Result
Returns the position (zero-based). If the specified index is out of range, this function returns 0 and DmGetLastErr returns an error code indicating the reason for failure. Note that this means a 0 return value might indicate either success or failure. If this function returns 0 and DmGetLastErr returns errNone, the return value indicates that this is the first record in the category.

Comments
Because this function must examine all records up to the current record, it can be slow to return, especially when called on a large database.

Records that have the deleted bit set are ignored, and if the user has specified that private records should be hidden or masked, private records are ignored as well.

If the record is ROM-based (pointer accessed) this routine makes a fake handle to it and stores this handle in the DmAccessType structure.

To learn which category a record is in, use DmRecordInfo.

See Also
DmQueryNextInCategory, DmSeekRecordInCategory, DmMoveCategory
**DmQueryNextInCategory**

**Purpose**
Return a handle to the next record in the specified category for reading only (does not set the busy bit).

**Declared In**
DataMgr.h

**Prototype**
```
MemHandle DmQueryNextInCategory (DmOpenRef dbP, UInt16 *indexP, UInt16 category)
```

**Parameters**
- `-> dbP` DmOpenRef to open database.
- `<> indexP` Index of a known record (often retrieved with `DmPositionInCategory`). If a “next” record is found, this index is updated to indicate that record.
- `> category` Index of category to query, or `dmAllCategories` to find the next record in any category.

**Result**
Returns a handle to the record following a known record, along with the index of that record. If a record couldn’t be found, this function returns `NULL`, and `DmGetLastErr` returns an error code indicating the reason for failure.

**Comments**
This function begins searching the database from the record at `*indexP` for a record that is in the specified category. If the record at `*indexP` belongs to that category, then a handle to it is returned. If not, the function continues searching until it finds a record in the category.

Records that have the `deleted` bit set are skipped, and if the user has specified that private records should be hidden or masked, private records are skipped as well.

Thus, if you want to find the next record in the category after the one you have an index for, increment the index value before calling this function. For example:
```
DmOpenRef myDB;
UInt16 record, attr, category, pos;
```
MemHandle newRecH;

DmRecordInfo(myDB, record, &attr, NULL, NULL);
category = attr & dmRecAttrCategoryMask;
pos = DmPositionInCategory(myDB, record, category);
pos++;
newRecH = DmQueryNextInCategory(myDB, &pos, category);

See Also  
DmNumRecordsInCategory, DmPositionInCategory, DmSeekRecordInCategory

DmQueryRecord

Purpose  
Return a handle to a record for reading only (does not set the busy bit).

Declared In  
DataMgr.h

Prototype  
MemHandle DmQueryRecord (DmOpenRef dbP, UInt16 index)

Parameters  
-> dbP  
DmOpenRef to open database.

-> index  
Which record to retrieve.

Result  
Returns a record handle. If an error occurs, this function returns NULL, and DmGetLastErr returns an error code indicating the reason for failure.

Some releases may display a fatal error message if the specified index is out of range.

Comments  
Returns a handle to the given record. Use this routine only when viewing the record. This routine successfully returns a handle to the record even if the record is busy.

If the record is ROM-based (pointer accessed) this routine returns the fake handle to it.
**DmQuickSort**

**Purpose**  
Sort records in a database.

**Declared In**  
DataMgr.h

**Prototype**  
Err DmQuickSort (DmOpenRef dbP, DmComparF *compar, Int16 other)

**Parameters**  
- `dbP`  
  Database access pointer.
- `compar`  
  Comparison function. See DmComparF.
- `other`  
  Any value the application wants to pass to the comparison function. This parameter is often used to indicate a sort direction (ascending or descending).

**Result**  
Returns errNone if no error, or one of the following if an error occurs:

- dmErrReadOnly
- dmErrNotRecordDB

Some releases may display a fatal error message instead of returning the error code.

**Comments**  
Deleted records are placed last in any order. All others are sorted according to the passed comparison function.

After DmQuickSort returns, equal database records do not have a consistent order. That is, if DmQuickSort is passed two equal records, their resulting order is unpredictable. To prevent records that contain the same data from being rearranged in an unpredictable order, pass the record’s unique ID to the comparison function (using the SortRecordInfoType structure).

DmQuickSort contains its own stack to limit uncontrolled recursion. When the stack is full DmQuickSort instead performs an insertion sort. An insertion sort is also performed when the number of records is low, avoiding the noticeable overhead of a quick sort with a small number of records. Finally, if the records seem mostly
sorted an insertion sort is performed to move only those records that need moving.

See Also  
DmFindSortPositionV10, DmInsertionSort

**DmRecordInfo**

**Purpose**  
Retrieve the record information as stored in the database header.

**Declared In**  
DataMgr.h

**Prototype**  
Err DmRecordInfo (DmOpenRef dbP, UInt16 index, Uint16 *attrP, Uint32 *uniqueIDP, LocalID *chunkIDP)

**Parameters**  
- `-> dbP`  
  DmOpenRef to open database.

- `-> index`  
  Index of the record.

- `<- attrP`  
  The record’s attributes. See “Record Attribute Constants.” Pass NULL for this parameter if you don’t want to retrieve this value.

- `<- uniqueIDP`  
  The record’s unique ID. Pass NULL for this parameter if you don’t want to retrieve this value.

- `<- chunkIDP`  
  The record’s local ID. Pass NULL for this parameter if you don’t want to retrieve this value.

**Result**  
Returns errNone if no error or dmErrIndexOutOfRange if the specified record can’t be found. Some releases may display a fatal error message instead of returning the error code.

See Also  
DmNumRecords, DmSetRecordInfo, DmQueryNextInCategory
### DmReleaseRecord

**Purpose**
Clear the busy bit for the given record and set the dirty bit if dirty is true.

**Declared In**
DataMgr.h

**Prototype**
```
Err DmReleaseRecord (DmOpenRef dbP, UInt16 index, Boolean dirty)
```

**Parameters**
- `dbP` DmOpenRef to open database.
- `index` The record to unlock.
- `dirty` If true, set the dirty bit.

**Result**
Returns errNone if no error, or `dmErrIndexOutOfRange` if the specified index is out of range. Some releases may display a fatal error message instead of returning the error code.

**Comments**
Call this routine when you finish modifying or reading a record that you’ve called `DmGetRecord` on or created using `DmNewRecord`.

**See Also**
DmGetRecord

### DmReleaseResource

**Purpose**
Release a resource acquired with `DmGetResource`.

**Declared In**
DataMgr.h

**Prototype**
```
Err DmReleaseResource (MemHandle resourceH)
```

**Parameters**
- `resourceH` Handle to resource.

**Result**
Returns errNone if no error.
DmRemoveRecord

Purpose
Remove a record from a database and dispose of its data chunk.

Declared In
DataMgr.h

Prototype
Err DmRemoveRecord (DmOpenRef dbP, UInt16 index)

Parameters
- > dbP DmOpenRef to open database.
- > index Index of the record to remove.

Result
Returns errNone if no error, or one of the following if an error occurs:
- dmErrReadOnly
- dmErrIndexOutOfRange
- dmErrNotRecordDB
- memErrChunkLocked
- memErrInvalidParam

Some releases may display a fatal error message instead of returning the error code.

Comments
Disposes of a the record’s data chunk and removes the record’s entry from the database header. DmRemoveRecord should only be used for newly-created records that have just been deleted or records that have never been sync’ed.

See Also
DmDetachRecord, DmDeleteRecord, DmArchiveRecord, DmNewRecord
DmRemoveResource

Purpose  Delete a resource from a resource database.

Declared In  DataMgr.h

Prototype  Err DmRemoveResource (DmOpenRef dbP, UInt16 index)

Parameters
- dbP  DmOpenRef to open database.
- index  Index of resource to delete.

Result  Returns errNone if no error, or one of the following if an error occurs:
- dmErrCorruptDatabase
- dmErrIndexOutOfRange
- dmErrReadOnly
- memErrChunkLocked
- memErrInvalidParam
- memErrNotEnoughSpace

May display a fatal error message if the database is not a resource database.

Comments  This routine disposes of the memory manager chunk that holds the given resource and removes its entry from the database header.

See Also  DmDetachResource, DmRemoveResource, DmAttachResource
DmRemoveSecretRecords

Purpose
Remove all secret records.

Declared In
DataMgr.h

Prototype
Err DmRemoveSecretRecords (DmOpenRef dbP)

Parameters
- dbP DmOpenRef to open database.

Result
Returns errNone if no error, or one of the following if an error occurs:
- dmErrReadOnly
- dmErrNotRecordDB
Some releases may display a fatal error message instead of returning the error code.

See Also
DmRemoveRecord, DmRecordInfo, DmSetRecordInfo

DmResizeRecord

Purpose
Resize a record by index.

Declared In
DataMgr.h

Prototype
MemHandle DmResizeRecord (DmOpenRef dbP, UInt16 index, UInt32 newSize)

Parameters
- dbP DmOpenRef to open database.
- index Which record to retrieve.
- newSize New size of record.

Result
Handle to resized record. Returns NULL if there is not enough space to resize the record, and DmGetLastErr returns an error code indicating the reason for failure. Some releases may display a fatal error message instead of returning the error code.
Comments
This routine reallocates the record in another heap of the same memory card if the current heap is not big enough. If this happens, the handle changes, so be sure to use the returned handle to access the resized record.

DmResizeResource

Purpose
Resize a resource and return the new handle.

Declared In
DataMgr.h

Prototype
MemHandle DmResizeResource (MemHandle resourceH, UInt32 newSize)

Parameters
- resourceH Handle to resource.
- newSize Desired new size of resource.

Result
Returns a handle to newly sized resource. Returns NULL if there is not enough space to resize the resource, and DmGetLastErr returns an error code indicating the reason for failure. Some releases may display a fatal error message instead of returning the error code.

Comments
Resizes the resource and returns a new handle. If necessary in order to grow the resource, this routine will reallocate it in another heap on the same memory card that it is currently in.

The handle may change if the resource had to be reallocated in a different data heap because there was not enough space in its present data heap.
DmResourceInfo

Purpose
Retrieve information on a given resource.

Declared In
DataMgr.h

Prototype
Err DmResourceInfo (DmOpenRef dbP, UInt16 index, DmResType *resTypeP, DmResID *resIDP, LocalID *chunkLocalIDP)

Parameters
-> dbP DmOpenRef to open database.
-> index Index of resource to get info on.
<- resTypeP The resource type. Pass NULL if you don’t want to retrieve this information.
<- resIDP The resource ID. Pass NULL if you don’t want to retrieve this information.
<- chunkLocalIDP The memory manager local ID of the resource data. Pass NULL if you don’t want to retrieve this information.

Result
Returns errNone if no error or dmErrIndexOutOfRange if an error occurred. May display a fatal error message if the database is not a resource database.

Comments
If dbP is a pointer to a base resource database, the information returned is about the resource from that database alone; this function ignores any associated overlay.

See Also
DmGetResource, DmGet1Resource, DmSetResourceInfo, DmFindResource, DmFindResourceType
**DmSearchRecord**

**Purpose**
Search all open record databases for a record with the handle passed.

**Declared In**
DataMgr.h

**Prototype**
Uint16 DmSearchRecord (MemHandle recH, DmOpenRef *dbPP)

**Parameters**
- `recH` Record handle.
- `dbPP` The database that contains the record `recH`.

**Result**
Returns the index of the record and database access pointer; if not found, returns -1 and `*dbPP` is 0.

**See Also**
DmGetRecord, DmFindRecordByID, DmRecordInfo

**DmSearchResource**

**Purpose**
Search all open resource databases for a resource by type and ID, or by pointer if it is non-NULL.

**Declared In**
DataMgr.h

**Prototype**
Uint16 DmSearchResource (DmResType resType, DmResID resID, MemHandle resH, DmOpenRef *dbPP)

**Parameters**
- `resType` Type of resource to search for.
- `resID` ID of resource to search for.
- `resH` Pointer to locked resource, or NULL.
- `dbPP` The resource database that contains the specified resource.

**Result**
Returns the index of the resource, stores DmOpenRef in `dbPP`. 
Comments This routine can be used to find a resource in all open resource databases by type and ID or by pointer. If \texttt{resH} is \texttt{NULL}, the resource is searched for by type and ID. If \texttt{resH} is not \texttt{NULL}, \texttt{resType} and \texttt{resID} is ignored and the index of the resource handle is returned. On return, \texttt{*dbPP} contains the access pointer of the resource database that the resource was eventually found in. Once the index of a resource is determined, it can be locked down and accessed by calling \texttt{DmGetResourceIndex}.

If any of the open databases are overlaid, this function finds and returns the localized version of the resource when searching by type and creator. In this case, the \texttt{dbPP} return value is a pointer to the overlay database, not the base resource database.

See Also \texttt{DmGetResource}, \texttt{DmFindResourceType}, \texttt{DmResourceInfo}, \texttt{DmFindResource}
DmSeekRecordInCategory

Purpose
Return the index of the record nearest the offset from the passed record index whose category matches the passed category. (The offset parameter indicates the number of records to move forward or backward.)

Declared In
DataMgr.h

Prototype
Err DmSeekRecordInCategory (DmOpenRef dbP, UInt16 *indexP, UInt16 offset, Int16 direction, UInt16 category)

Parameters
- dbP: DmOpenRef to open database.
- index: The index to start the search at. Upon return, contains the index of the record at offset from the index that you passed in.
- offset: Offset of the passed record index. This must be a positive number; use dmSeekBackward for the direction parameter to search backwards.
- direction: Must be either dmSeekForward or dmSeekBackward.
- category: Category index.

Result
Returns errNone if no error; returns dmErrIndexOutOfRange or dmErrSeekFailed if an error occurred.

Comments
DmSeekRecordInCategory searches for a record in the specified category. The search begins with the record at index. When it finds a record in the specified category, it decrements the offset parameter and continues searching until a match is found and offset is 0.

Because of this, if you use DmSeekRecordInCategory to find the nearest matching record in a particular category, you must pass different offset parameters if the starting record is in the category than if it isn’t. If the record at index is in the category, then you
must pass an offset of 1 to find the next record in the category because the comparison is performed before the index value changes. If the record at index isn’t in the category, you must pass an offset of 0 to find the next record in the category. In this case, an offset of 1 skips the first matching record.

Records that have the deleted bit set are ignored, and if the user has specified that private records should be hidden or masked, private records are ignored as well.

See Also  
DmNumRecordsInCategory, DmQueryNextInCategory, DmPositionInCategory, DmMoveCategory

DmSet

Purpose  
Write a specified value into a section of a record. This function also checks the validity of the pointer for the record and makes sure the writing of the record information doesn’t exceed the bounds of the record.

Declared In  
DataMgr.h

Prototype  
Err DmSet (void *recordP, UInt32 offset, UInt32 bytes, UInt8 value)

Parameters  
- > recordP  
  Pointer to locked data record (chunk pointer).
- > offset  
  Offset within record to start writing.
- > bytes  
  Number of bytes to write.
- > value  
  Byte value to write.

Result  
Returns errNone if no error. May display a fatal error message if the record pointer is invalid or the function overwrites the record.

Comments  
Must be used to write to data manager records because the data storage area is write-protected.

See Also  
DmWrite
DmSetDatabaseInfo

Purpose
Set information about a database.

Declared In
DataMgr.h

Prototype
Err DmSetDatabaseInfo (UInt16 cardNo,
LocalID dbID, const Char *nameP,
UInt16 *attributesP, UInt16 *versionP,
UInt32 *crDateP, UInt32 *modDateP,
UInt32 *bckUpDateP, UInt32 *modNumP,
LocalID *appInfoIDP, LocalID *sortInfoIDP,
UInt32 *typeP, UInt32 *creatorP)

Parameters
-> cardNo Card number the database resides on.
-> dbID Database ID of the database.
-> nameP Pointer to the new name of the database, or NULL. A database name can be up to 32 ASCII bytes long, including the null terminator (as specified by dmDBNameLength). Database names must use only 7-bit ASCII characters (0x20 through 0x7E).
-> attributesP Pointer to new attributes variable, or NULL. See “Database Attribute Constants” for a list of possible values.
-> versionP Pointer to new version, or NULL.
-> crDateP Pointer to new creation date variable, or NULL. Specify the value as a number of seconds since Jan. 1, 1904.
-> modDateP Pointer to new modification date variable, or NULL. Specify the value as a number of seconds since Jan. 1, 1904.
-> bckUpDateP Pointer to new backup date variable, or NULL. Specify the value as a number of seconds since Jan. 1, 1904.
Data and Resource Manager
Data Manager Functions

-> modNumP  Pointer to new modification number variable, or NULL.
-> appInfoIDP  Pointer to new appInfoID variable, or NULL.
-> sortInfoIDP  Pointer to new sortInfoID variable, or NULL.
-> typeP  Pointer to new type variable, or NULL.
-> creatorP  Pointer to new creator variable, or NULL.

Result  Returns errNone if no error or one of the following if an error occurred:
  • dmErrInvalidDatabaseName
  • dmErrAlreadyExists
  • dmErrInvalidParam

Comments  When this call changes appInfoID or sortInfoID, the old chunk ID (if any) is marked as an orphaned chunk1 and the new chunk ID is unorphaned. Consequently, you shouldn’t replace an existing appInfoID or sortInfoID if that chunk has already been attached to another database.

Call this routine to set any or all information about a database except for the card number and database ID. This routine sets the new value for any non-NULL parameter.

See Also  DmDatabaseInfo, DmOpenDatabaseInfo, DmFindDatabase, DmGetNextDatabaseByTypeCreator, TimDateTimeToSeconds

1. An “orphaned chunk” is one that is allocated in the storage heap, but to which nothing refers. If the orphaned chunk is not put into a database as a record, an appInfo block, or the like, and if the application doesn’t keep track of it—in a global variable, perhaps—it could get lost. If the application doesn’t get around to freeing the chunk before it quits or crashes, or before the device is reset, that storage will be forever unusable: the user can’t delete it since the user only deletes databases.

During a soft reset, the OS walks through the storage heap and frees any orphaned chunks that it finds. Since most users reset only rarely, however, you shouldn’t rely on this happening.
**DmSetRecordInfo**

**Purpose**
Set record information stored in the database header.

**Declared In**
DataMgr.h

**Prototype**
Err DmSetRecordInfo (DmOpenRef dbP, UInt16 index, UInt16 *attrP, UInt32 *uniqueIDP)

**Parameters**
- dbP: DmOpenRef to open database.
- index: Index of record.
- attrP: Pointer to new attribute variable, or NULL. See “Record Attribute Constants” for a list of possible values.
- uniqueIDP: Pointer to new unique ID variable, or NULL.

**Result**
Returns errNone if no error, or one of the following if an error occurred:
- dmErrReadOnly
- dmErrNotRecordDB
- dmErrIndexOutOfRange

Some releases may display a fatal error message instead of returning the error code.

**Comments**
Sets information about a record. This routine cannot be used to set the dmRecAttrBusy bit; instead, use DmGetRecord to set the bit and DmReleaseRecord to clear it.

Normally, the unique ID for a record is automatically created by the data manager when a record is created using DmNewRecord, so an application would not typically change the unique ID.

**See Also**
DmNumRecords, DmRecordInfo
**DmSetResourceInfo**

**Purpose**
Set information on a given resource.

**Declared In**
DataMgr.h

**Prototype**
Err DmSetResourceInfo (DmOpenRef dbP, 
UInt16 index, DmResType *resTypeP,
DmResID *resIDP)

**Parameters**
- -> dbP DmOpenRef to open database.
- -> index Index of resource to set info for.
- -> resTypeP Pointer to new resType (resource type), or NULL.
- -> resIDP Pointer to new resource ID, or NULL.

**Result**
Returns errNone if no error, or one of the following if an error occurred:
- dmErrIndexOutOfRange
- dmErrReadOnly

May display a fatal error message if the database is not a resource database.

**Comments**
Use this routine to set all or a portion of the information on a particular resource. Any or all of the new info pointers can be NULL. If not NULL, the type and ID of the resource are changed to *resTypeP and *resIDP.
DmStrCopy

Purpose
Copies a string to a record within a database that is open for writing.

Declared In
DataMgr.h

Prototype
Err DmStrCopy (void *recordP, UInt32 offset, const Char *srcP)
**Parameters**

`<-> recordP`  
Pointer to data record (chunk pointer).

`-> offset`  
Offset within record to start writing.

`-> srcP`  
Pointer to null-terminated string.

**Result**

Returns `errNone` if no error. May display a fatal error message if the record pointer is invalid or the function overwrites the record.

**Comments**

This is one of the routines that must be used to write to Data Manager records; because the data storage area is write-protected, you cannot write to it directly. This routine checks the validity of the chunk pointer for the record to insure that writing the record will not exceed the chunk bounds. `DmStrCopy` is a convenience method that determines the size of the supplied string and then simply calls `DmWrite`.

**See Also**

`DmSet`

---

**DmWrite**

**Purpose**

Copies a specified number of bytes to a record within a database that is open for writing.

**Declared In**

`DataMgr.h`

**Prototype**

```c
Err DmWrite (void *recordP, UInt32 offset, const void *srcP, UInt32 bytes)
```

**Parameters**

`<-> recordP`  
Pointer to locked data record (chunk pointer).

`-> offset`  
Offset within record to start writing.

`-> srcP`  
Pointer to data to copy into record.

`-> bytes`  
Number of bytes to write.

**Result**

Returns `errNone` if no error. May display a fatal error message if the record pointer is invalid or the function overwrites the record.
Comments
This is one of the routines that must be used to write to Data Manager records; because the data storage area is write-protected, you cannot write to it directly. This routine checks the validity of the chunk pointer for the record to insure that writing the record will not exceed the chunk bounds.

See Also
DmStrCopy, DmSet

DmWriteCheck

Purpose
Check the parameters of a write operation to a data storage chunk before actually performing the write.

Declared In
DataMgr.h

Prototype
Err DmWriteCheck (void *recordP, UInt32 offset, UInt32 bytes)

Parameters
- recordP  Locked pointer to recordH.
- offset  Offset into record to start writing.
- bytes  Number of bytes to write.

Result
Returns errNone if no error; returns dmErrNotValidRecord or dmErrWriteOutOfBounds if an error occurred.
Application-Defined Functions

DmComparF

Purpose
Comparing two records in a database.

Declared In
DataMgr.h

Prototype
typedef Int16 DmComparF (void *rec1, void *rec2,
Int16 other, SortRecordInfoPtr rec1SortInfo,
SortRecordInfoPtr rec2SortInfo,
MemHandle appInfoH)

Parameters
- rec1, rec2
  Pointers to the two records to compare.
- other
  Any other custom information you want passed to the comparison function. This parameter is often used to indicate a sort direction (ascending or descending).
- rec1SortInfo, rec2SortInfo
  Pointers to SortRecordInfoType structures that specify unique sorting information for the two records.
- appInfoH
  A handle to the database’s application info block.

Result
Returns:
- 0 if rec1 = rec2.
- < 0 if rec1 < rec2.
- > 0 if rec1 > rec2.

Comments
This function is used to sort the records in a database. It is specifically called by DmFindSortPosition, DmInsertionSort, and DmQuickSort.

Compatibility
The DmComparF prototype changed in Palm OS version 2.0. Previously, the prototype only defined the first three parameters.
As a rule, the change in the number of arguments from three to six doesn’t cause problems when a 1.0 application is run on a 2.0 device because the system only pulls the arguments from the stack that are there.

Keep in mind, however, that some optimized applications built with tools other than Metrowerks CodeWarrior for Palm OS may have problems as a result of the change in arguments when running on a 2.0 or later device.
Error Manager

This chapter provides reference material for the error manager. The error manager API is declared in the header files ErrorMgr.h and ErrorBase.h. This chapter covers:

- **ERROR_CHECK_LEVEL Define**
- **Error Manager Data Structures**
- **Error Manager Functions**

For more information on the error manager, see the chapter “Debugging Strategies” in the *Palm OS Programmer’s Companion*, vol. I.

**ERROR_CHECK_LEVEL Define**

The error manager uses the compiler define **ERROR_CHECK_LEVEL** to control the level of error messages displayed. You can set the value of the compiler define to control which level of error checking and display is compiled into the application. Three levels of error checking are supported: none, partial, and full.

<table>
<thead>
<tr>
<th>If you set <strong>ERR_CHECK_LEVEL</strong> to...</th>
<th>The compiler...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_CHECK_NONE (0)</td>
<td>Doesn’t compile in any error calls.</td>
</tr>
<tr>
<td>ERROR_CHECK_PARTIAL (1)</td>
<td>Compiles in only ErrDisplay and ErrFatalDisplayIf calls.</td>
</tr>
<tr>
<td>ERROR_CHECK_FULL (2)</td>
<td>Compiles in all three calls.</td>
</tr>
</tbody>
</table>
## Error Manager Data Structures

### ErrExceptionType

An `ErrExceptionType` structure is created for each `ErrTry` and `ErrCatch` block. At any point in the program, there is a linked list of these structures. An `ErrExceptionType` structure stores information about the state of the machine (register values) at the start of the `ErrTry` block.

```c
typedef struct ErrExceptionType {
   struct ErrExceptionType *nextP;
   ErrJumpBuf state;
   Int32 err;
} ErrExceptionType;

typedef ErrExceptionType *ErrExceptionPtr;
```

### Field Descriptions

- **nextP**: Next `ErrExceptionType` structure in the linked list.
- **state**: Storage for `setjmp/longjmp`.
- **err**: Error code.

## Error Manager Functions

### ErrAlert

**Purpose**

Macro that displays an alert dialog for runtime errors.

**Declared In**

`ErrorBase.h`

**Prototype**

```c
ErrAlert (err)
```

**Parameters**

- `err`: An error code (as type `Err`).

**Result**

Returns 0, which indicates that the OK button has been clicked to dismiss the dialog.
Comments  
This macro is intended for use by applications that are likely to receive runtime errors when the application itself is not at fault. For example, a networking application might use it to display an alert if the remote server cannot be found.

The error message displayed on the dialog is stored in a 'tSTL' resource. A 'tSTL' resource contains a list of strings that can be looked up by index. The err parameter is used as the index into this list.

To use application-defined error codes in ErrAlert, make sure that all of your error codes are greater than or equal to appErrorClass. This way, the error manager looks up the code in the application’s 'tSTL' resource number 0. All other error codes are taken from 'tSTL' resource stored in the system.

Compatibility  
Implemented only if 3.2 New Feature Set is present.

ErrCatch

Purpose  
Macro that marks the end of an ErrTry block and the beginning of an ErrCatch block.

Declared In  
ErrorBase.h

Prototype  
ErrCatch (theErr)

Parameters  
-> theErr  
An exception code identifying the reason for the failure. This is the value supplied to the ErrThrow call that caused the jump to this ErrCatch block.

Result  
Returns nothing.

Comments  
ErrCatch can only be used in conjunction with ErrTry and ErrEndCatch. See the comments under ErrTry for usage instructions.

ErrTry, ErrCatch and ErrThrow are based on setjmp and longjmp. At the beginning of an ErrTry block, setjmp saves the
machine registers. `ErrThrow` calls `longjmp`, which restores the
registers and jumps to the beginning of the `ErrCatch` block.
Therefore, any changes in the `ErrTry` block to variables stored in
registers aren’t retained when entering the `ErrCatch` block.

The solution is to declare variables that you want to use in both the
`ErrTry` and `ErrCatch` blocks as “volatile”. For example:

```c
volatile long x = 1;  // Declare volatile local variable
ErrTry {
    x = 100;    // Set local variable in Try
    ErrThrow(-1);
}
ErrCatch(inErr) {
    if (x > 1) {  // Use local variable in Catch
        SysBeep(1);
    }
} ErrEndCatch
```

If you have many local variables after the `ErrCatch` you may want
to put the `ErrTry` and `ErrCatch` in a separate enclosing function.

### ErrDisplay

**Purpose**  
Macro that displays an error alert if error checking is set to partial or
full.

**Declared In**  
`ErrorMgr.h`

**Prototype**  
`ErrDisplay (msg)`

**Parameters**  
`-> msg`  
Error message text as a string.

**Result**  
No return value.

**Comments**  
Call this macro to display an error message, source code filename,
and line number. This macro is compiled into the code only if the
compiler define `ERROR_CHECK_LEVEL` is set to 1 or 2
(`ERROR_CHECK_PARTIAL` or `ERROR_CHECK_FULL`).

**See Also**  
`ErrFatalDisplayIf`, `ErrNonFatalDisplayIf`
**ErrDisplayFileLineMsg**

**Purpose**
Display a dialog with an error message. Do not allow the user to exit the dialog or continue.

**Declared In**
ErrorBase.h

**Prototype**
void ErrDisplayFileLineMsg(const Char *const filename, Uint16 lineno, const Char *const msg)

**Parameters**
- `filename` Source code filename.
- `lineno` Line number in the source code file.
- `msg` Message to display.

**Result**
Never returns.

**Comment**
Called by **ErrFatalDisplayIf** and **ErrNonFatalDisplayIf**. This function is useful when the application is already on the device and being tested by users.

On Japanese systems, the system displays a generic message indicating that an error has occurred instead of displaying the English message.

**See Also**
ErrFatalDisplayIf, ErrNonFatalDisplayIf, ErrDisplay

**ErrEndCatch**

**Purpose**
Macro that marks the end of an **ErrCatch** block.

**Declared In**
ErrorBase.h

**Prototype**
ErrEndCatch

**Parameters**
None.

**Result**
Returns nothing.
Comments  ErrEndCatch can only be used in conjunction with ErrTry and ErrCatch. See the comments under ErrTry for usage instructions.

ErrExceptionList

Purpose  Return the address of the pointer to the first ErrExceptionType structure in the linked list of exception structures.

Declared In  ErrorBase.h

Prototype  MemPtr *ErrExceptionList (void)

Parameters  None.

Result  Returns the address of the pointer to the first ErrExceptionType structure linked into the exception list.

Comments  This function is used by the ErrTry and ErrCatch macros as well as the ErrThrow function in order to find the exception list.

When called from an application, this routine returns the application’s exception list.

ErrFatalDisplayIf

Purpose  Macro that displays an error alert dialog if condition is true and error checking is set to partial or full.

Declared In  ErrorMgr.h

Prototype  ErrFatalDisplayIf (condition, msg)

Parameters  -> condition  A boolean value. If true, display the error.
             -> msg  Error message text as a string.

Result  No return value.
Comments
Call this macro to display a fatal error message, source code filename, and line number. The alert is displayed only if condition is true. The dialog is cleared only when the user resets the system by responding to the dialog.

This macro is compiled into the code if the compiler define ERROR_CHECK_LEVEL is set to 1 or 2 (ERROR_CHECK_PARTIAL or ERROR_CHECK_FULL).

See Also ErrNonFatalDisplayIf, ErrDisplay

ErrNonFatalDisplayIf

Purpose
Macro that displays an error alert dialog if condition is true and error checking is set to full.

Declared In ErrorMgr.h

Prototype
ErrNonFatalDisplayIf (condition, msg)

Parameters
-> condition A boolean value. If true, display the error.
-> msg Error message text as a string.

Result
No return value.

Comments
Call this macro to display a nonfatal error message, source code filename, and line number. The alert is displayed only if condition is true. The alert dialog is cleared when the user selects to continue (or resets the system).

This macro is compiled into the code only if the compiler define ERROR_CHECK_LEVEL is set to 2 (ERROR_CHECK_FULL).

See Also ErrFatalDisplayIf, ErrDisplay
**ErrThrow**

**Purpose**
Cause a jump to the nearest Catch block.

**Declared In**
ErrorBase.h

**Prototype**
void ErrThrow (Int32 err)

**Parameters**
-> err Error code.

**Result**
Never returns.

**Comments**
Use the macros ErrTry, ErrCatch, and ErrEndCatch in conjunction with this function.

**See Also**
ErrFatalDisplayIf, ErrNonFatalDisplayIf, ErrDisplay

---

**ErrTry**

**Purpose**
Macro that marks the beginning of a try/catch block.

**Declared In**
ErrorBase.h

**Prototype**
ErrTry

**Parameters**
None.

**Result**
Returns nothing.

**Comments**
An exception raised by a call to **ErrThrow**—even from within a nested subroutine—causes program execution to switch to the beginning of the **ErrCatch** block. If the end of the block enclosed by **ErrTry** is encountered without a call to **ErrThrow**, execution jumps to the line of code following the **ErrEndCatch** macro. See “The Try-and-Catch Mechanism” on page 377 of the Palm OS Programmer’s Companion, vol. I for a more thorough description of how this mechanism works.
Example

You must structure your code exactly as shown here. You can’t use ErrTry without ErrCatch and ErrEndCatch, or vice versa.

ErrTry {
    // Do something which may fail. Call ErrThrow to signal failure and force a jump to the following ErrCatch block.
}
ErrCatch(inErr) {
    // Recover or cleanup after a failure in the above ErrTry block. "inErr" is an exception code identifying the reason for the failure.
    // Call ErrThrow if you want to jump out to the next ErrCatch block.
    // The code in this block doesn’t execute if the above ErrTry block completes without a call to ErrThrow.
} ErrEndCatch
Expansion Manager

This chapter provides the following information about the Expansion Manager:

- Expansion Manager Data Structures
- Expansion Manager Constants
- Expansion Manager Functions

The header file `ExpansionMgr.h` declares the Expansion Manager API. For more information on the Expansion Manager, see Chapter 7, “Expansion,” in *Palm OS Programmer’s Companion*, vol. I.

Note that the Expansion Manager is an optional system extension; the functions described in this chapter are implemented only if the Expansion Manager Feature Set is present.

Expansion Manager Data Structures

ExpCardInfoType

The `ExpCardInfoType` declaration defines a structure that is passed to `ExpCardInfo`. This structure is used to determine the characteristics of the card loaded in the slot. It is initialized by the underlying slot driver with the following information.

```c
typedef struct ExpCardInfoTag {
    UInt32 capabilityFlags;
    Char manufacturerStr[expCardInfoStringMaxLen+1];
    Char productStr[expCardInfoStringMaxLen+1];
    Char deviceClassStr[expCardInfoStringMaxLen+1];
    Char deviceUniqueIDStr[expCardInfoStringMaxLen+1];
} ExpCardInfoType, *ExpCardInfoPtr;
```
### Expansion Manager Constants

#### Field Descriptions

- **capabilityFlags**: Describes the capabilities of the card. The following flags are currently supported:
  - `expCapabilityHasStorage` indicates that the card supports reading and (possibly) writing.
  - `expCapabilityReadOnly` indicates that the card is read only.
  - `expCapabilitySerial` indicates that the card supports a simple serial interface.

- **manufacturerStr**: Names the manufacturer of the card. For example “Palm” or “Motorola”.

- **productStr**: Name of the product. For example “SafeBackup 32MB”.

- **deviceClassStr**: Describes the type of card, for example, “Backup” or “Ethernet”.

- **deviceUniqueIDStr**: Unique identifier for the product. A serial number for example. This value is set to the empty string if no identifier exists.

### Expansion Manager Constants

#### Error Codes

The Expansion Manager defines the following error codes:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>expErrUnsupportedOperation</code></td>
<td>The operation is unsupported or undefined.</td>
</tr>
<tr>
<td><code>expErrNotEnoughPower</code></td>
<td>The required power is not available.</td>
</tr>
<tr>
<td><code>expErrCardNotPresent</code></td>
<td>There is no card present in the given slot.</td>
</tr>
</tbody>
</table>
## Defined Media Types

The following media types are defined by the Expansion Manager. These media types are used with the function `VFSVolumeInfo` in the `VolumeInfoType.mediaType` field. The media type is also passed as a parameter to the `VFSRegisterDefaultDirectory` and `VFSUnregisterDefaultDirectory` functions.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>expErrInvalidSlotRefNum</code></td>
<td>The slot reference number is not valid.</td>
</tr>
<tr>
<td><code>expErrSlotDeallocated</code></td>
<td>The slot reference number is within the valid range, but the slot has been deallocated.</td>
</tr>
<tr>
<td><code>expErrCardNoSectorReadWrite</code></td>
<td>The card does not support the slot driver block read/write API.</td>
</tr>
<tr>
<td><code>expErrCardReadOnly</code></td>
<td>The card supports the slot driver block read/write API but the card is read only.</td>
</tr>
<tr>
<td><code>expErrCardBadSector</code></td>
<td>The card supports the slot driver block read/write API but the sector is bad.</td>
</tr>
<tr>
<td><code>expErrCardProtectedSector</code></td>
<td>The card supports the slot driver block read/write API but the sector is protected.</td>
</tr>
<tr>
<td><code>expErrNotOpen</code></td>
<td>The slot driver library has not been opened.</td>
</tr>
<tr>
<td><code>expErrStillOpen</code></td>
<td>The slot driver library is still open; it may have been opened more than once.</td>
</tr>
<tr>
<td><code>expErrUnimplemented</code></td>
<td>The call is unimplemented.</td>
</tr>
<tr>
<td><code>expErrEnumerationEmpty</code></td>
<td>There are no values remaining to enumerate.</td>
</tr>
<tr>
<td><code>expErrIncompatibleAPIVer</code></td>
<td>The API version of the underlying slot driver is not supported by this version of Expansion Manager.</td>
</tr>
</tbody>
</table>
Expansion Manager
Expansion Manager Functions

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expMediaType_Any</td>
<td>'wild'</td>
<td>Matches all media types when looking up a default directory</td>
</tr>
<tr>
<td>expMediaType_MemoryStick</td>
<td>'mstk'</td>
<td>Memory stick</td>
</tr>
<tr>
<td>expMediaType_CompactFlash</td>
<td>'cfsh'</td>
<td>Compact Flash</td>
</tr>
<tr>
<td>expMediaType_SecureDigital</td>
<td>'sdig'</td>
<td>Secure Digital</td>
</tr>
<tr>
<td>expMediaType_MultiMediaCard</td>
<td>'mmcd'</td>
<td>MultiMedia Card</td>
</tr>
<tr>
<td>expMediaType_SmartMedia</td>
<td>'smed'</td>
<td>SmartMedia</td>
</tr>
<tr>
<td>expMediaType_RAMDisk</td>
<td>'ramd'</td>
<td>A RAM disk based media</td>
</tr>
<tr>
<td>expMediaType_PoserHost</td>
<td>'pose'</td>
<td>Host file system emulated by the Palm OS® Emulator</td>
</tr>
<tr>
<td>expMediaType_MacSim</td>
<td>'PSim'</td>
<td>Host file system emulated by the Mac Simulator</td>
</tr>
</tbody>
</table>

ExpCardGetSerialPort

**Purpose**
Get a card’s serial port creator ID for use in serial access.

**Declared In**
ExpansionMgr.h

**Prototype**
Err ExpCardGetSerialPort(UInt16 slotRefNum, UInt32 *portP)

**Parameters**
-> slotRefNum Slot number of slot to check.
<- portP  Pointer to UInt32 into which the serial port creator ID is stored.

**Result**

Returns the following result codes:

- **errNone**  No error
- **expErrInvalidSlotRefNum**  The specified slot number is invalid.
- **expErrSlotDeallocated**  The specified slot number is within the valid range but has been deallocated.

**Compatibility**

Implemented only if the [Expansion Manager Feature Set](#) is present.

**See Also**  [ExpCardInfo](#)

---

**ExpCardInfo**

**Purpose**

Obtains information about a card in a given slot.

**Declared In**  ExpansionMgr.h

**Prototype**

```c
Err ExpCardInfo(UInt16 slotRefNum, ExpCardInfoType *infoP)
```

**Parameters**

- **-> slotRefNum**  Slot number.
- **<- infoP**  Pointer to [ExpCardInfoType](#) structure.

**Result**

Returns one of the following result codes:

- **errNone**  No error
- **expErrCardNotPresent**  There is no card present in the specified slot.
- **expErrInvalidSlotRefNum**  The slot number is invalid.
expErrSlotDeallocated
The slot number is within the valid range but has been deallocated.

Comments This function returns information about a card, including whether the card supports secondary storage or is strictly read-only, by filling in the ExpCardInfoType structure’s fields.

Compatibility Implemented only if the Expansion Manager Feature Set is present.

See Also ExpCardGetSerialPort, ExpCardPresent, ExpSlotEnumerate

ExpCardPresent

Purpose Determines if a card is present in the given slot.

Declared In ExpansionMgr.h

Prototype Err ExpCardPresent(UInt16 slotRefNum)

Parameters -> slotRefNum Slot number.

Result Returns the following result codes:

errNone A card is present in the specified slot.
expErrCardNotPresent There is no card present in the specified slot.
expErrInvalidSlotRefNum The specified slot number is not valid.
expErrSlotDeallocated The specified slot number is within the valid range but has been deallocated.
Comments  The Expansion Manager passes the call through to the appropriate slot driver.

Compatibility  Implemented only if the Expansion Manager Feature Set is present.

See Also  ExpCardInfo, ExpSlotEnumerate

**ExpSlotDriverInstall**

**Purpose**  Installs and initializes a slot driver shared library into the system table.

**Declared In**  ExpansionMgr.h

**Prototype**  
```
Err ExpSlotDriverInstall (UInt32 dbCreator, 
    UInt16 *slotLibRefNumP)
```

**Parameters**

- `-> dbCreator`  Database creator code of the slot driver library to be installed.
- `-> slotLibRefNumP`  Pointer to variable for returning the library reference number (on failure, `sysInvalidRefNum` is returned in this variable).

**Result**  Returns `errNone` if the slot driver is installed correctly. Because this function uses `SysLibInstall` to install the slot driver shared library, `ExpSlotDriverInstall` may return any of the error codes that `SysLibInstall` returns, including `sysErrLibNotFound`, `sysErrNoFreeRAM`, and `sysErrNoFreeLibSlots`. It may also return any error code returned by `SlotOpen`, the implementation of which is specific to a given device manufacturer.

**Comments**  This function is not typically called by applications but can be used to load additional slot drivers after the device has booted. It is called internally by the Expansion Manager to install a slot driver shared library into the library table and initialize it for use. Once installed,
the slotLibRefNum can be used by other functions to refer to the library.

**Compatibility**
Implemented only if the Expansion Manager Feature Set is present.

**See Also**
ExpSlotDriverRemove, ExpSlotLibFind

---

### ExpSlotDriverRemove

#### Purpose
Closes and remove a slot driver shared library from the system table.

#### Declared In
ExpansionMgr.h

#### Prototype
`Err ExpSlotDriverRemove (UInt16 slotLibRefNum)`

#### Parameters
- `-> slotLibRefNum`
  Slot driver shared library reference number.

#### Result
Returns `errNone`.

#### Comments
This function is not typically called by applications but can be used to unload slot drivers associated with external slots. It is called internally by the Expansion Manager to remove the shared library from the system table, and, if appropriate, release the slot allocated to the given `slotLibRefNum`. Prior to removing the slot driver, it unmounts any volumes associated with the slot.

**Compatibility**
Implemented only if the Expansion Manager Feature Set is present.

**See Also**
ExpSlotDriverInstall, ExpSlotLibFind, SysLibRemove
ExpSlotEnumerate

**Purpose**
Iterates through valid slot numbers.

**Declared In**
ExpansionMgr.h

**Prototype**
```
Err ExpSlotEnumerate(UInt16 *slotRefNumP,
                     UInt32 *slotIteratorP)
```

**Parameters**
- `<- slotRefNumP` Reference number of the currently-enumerated slot.
- `-> slotIteratorP` Pointer to the index of the last entry enumerated. For the first iteration, initialize this parameter to the constant `expIteratorStart`. Upon return this references the next entry in the directory. If this is the last entry, this parameter is set to `expIteratorStop`.

**Result**
Returns one of the following result codes:
- `errNone` The slot reference number indicated by `slotRefNumP` is valid.
- `expErrEnumerationEmpty` There are no slots left to enumerate. `slotRefNumP` is set to the slot number of the currently enumerated slot or `invalidSlotRefNum` if there are no slots.

**Comments**
This function iterates through the device’s slots. The first time this function is called, set `*slotIteratorP` to `expIteratorStart` to find the initial slot. Once set this value is changed with each subsequent call to this function until it reaches the maximum number of slots, at which point `ExpSlotEnumerate` sets `*slotIteratorP` to `expIteratorStop`.

**Example**
The following is an example of how `ExpSlotEnumerate` should be used:
**Expansion Manager**

**Expansion Manager Functions**

```c
UInt16 slotRefNum;
UInt32 slotIterator = expIteratorStart;
while (slotIterator != expIteratorStop) {
    if ((err = ExpSlotEnumerate(&slotRefNum,
                                &slotIterator)) == errNone) {
        // do something with the slotRefNum
    } else {
        // handle error (by breaking out of the
        // loop, most likely)
    }
}
```

**Compatibility**
Implemented only if the Expansion Manager Feature Set is present.

**See Also** ExpCardInfo, ExpCardPresent, ExpSlotDriverRemove

**ExpSlotLibFind**

**Purpose** Retrieves the slot driver library reference number for the driver that controls the specified slot.

**Declared In** ExpansionMgr.h

**Prototype**

```c
Err ExpSlotLibFind(UInt16 slotRefNum,
                   UInt16 *slotLibRefNum)
```

**Parameters**

- `-> slotRefNum` Slot number.
- `<- slotLibRefNum` Pointer to the reference number for the slot driver library allocated to the given slot.

**Result** Returns the following result codes:

- `errNone` No error.
- `expErrInvalidSlotRefNum` The slot number is invalid.
expErrSlotDeallocated
The slot number is within the valid range but has been deallocated.

Comments
This function returns the reference number to the slot driver library loaded for the given slotRefNum. This function is used when making calls directly to the slot driver library.

Compatibility
Implemented only if the Expansion Manager Feature Set is present.

See Also
ExpSlotDriverInstall, ExpSlotDriverRemove
Expansion Manager
Expansion Manager Functions
Feature Manager

This chapter provides reference material for the feature manager. The feature manager API is declared in the header file FeatureMgr.h.

For more information on the feature manager, see the section “Features” in the Palm OS Programmer’s Companion, vol. I.

To learn how to use the predefined Palm OS® features to test for the existence of certain OS features, see the “Compatibility Guide” appendix.

Feature Manager Functions

FtrGet

Purpose
Get a feature.

Declared In
FeatureMgr.h

Prototype
Err FtrGet (UInt32 creator, UInt16 featureNum, UInt32 *valueP)

Parameters
-> creator Creator ID, which must be registered with PalmSource, Inc. This is usually the same as the creator ID for the application that owns this feature.

-> featureNum Feature number of the feature.

<- valueP Value of the feature is returned here.

Result
Returns 0 if no error, or ftrErrNoSuchFeature if the specified feature number doesn’t exist for the specified creator.
Comments  The value of the feature is application-dependent.

See Also  FtrSet

FtrGetByIndex

Purpose  Get a feature by index.

Declared In  FeatureMgr.h

Prototype  Err FtrGetByIndex (UInt16 index, Boolean romTable, UInt32 *creatorP, UInt16 *numP, UInt32 *valueP)

Parameters  
- index  Index of feature.
- romTable  If true, index into ROM table; otherwise, index into RAM table.
- creatorP  Feature creator is returned here.
- numP  Feature number is returned here.
- valueP  Feature value is returned here.

Result  Returns 0 if no error, or ftrErrNoSuchFeature if the index is out of range.

Comments  This function is intended for system use only. It is used by shell commands. Most applications don’t need it.

Until the caller gets back ftrErrNoSuchFeature, it should pass indices for each table (ROM, RAM) starting at 0 and incrementing. Note that in Palm OS 3.1 and higher, the RAM feature table serves the entire system. At system startup, the values in the ROM feature table are copied into the RAM feature table.
**FtrPtrFree**

**Purpose**
Release memory previous allocated with FtrPtrNew.

**Declared In**
FeatureMgr.h

**Prototype**
Err FtrPtrFree (UInt32 creator, UInt16 featureNum)

**Parameters**
- creator
  The creator ID for the feature.
- featureNum
  Feature number of the feature.

**Result**
Returns 0 if no error, or ftrErrNoSuchFeature if an error occurs.

**Comments**
This function unregisters the feature before freeing the memory associated with it.

**Compatibility**
Implemented only if 3.1 New Feature Set is present.

**FtrPtrNew**

**Purpose**
Allocate feature memory.

**Declared In**
FeatureMgr.h

**Prototype**
Err FtrPtrNew (UInt32 creator, UInt16 featureNum, UInt32 size, void **newPtrP)

**Parameters**
- creator
  Creator ID, which must be registered with PalmSource, Inc. This is usually the same as the creator ID for the application that owns this feature.
- featureNum
  Feature number of the feature.
- size
  Size in bytes of the temporary memory to allocate. The maximum chunk size is 64K.
<- newPtrP  

**Result**  
Returns 0 if no error, memErrInvalidParam if the value of size is 0, or memErrNotEnoughSpace if there is not enough space to allocate a chunk of the specified size.

**Comments**  
This function allocates a chunk of memory and stores a pointer to that chunk in the feature table. The same pointer is returned in newPtrP. The memory chunk remains allocated and locked until the next system reset or until you free the chunk with FtrPtrFree.

FtrPtrNew is useful if you want quick, efficient access to data that persists from one invocation of the application to the next. FtrPtrNew stores values on the storage heap rather than the dynamic heap, where free space is often extremely limited. The disadvantage to using feature memory is that writing to storage memory is slower than writing to dynamic memory.

**NOTE:** Starting with Palm OS 3.5 FtrPtrNew allows allocating chunks larger than 64k. Do keep in mind standard issues with allocating large chunks of memory: there might not be enough contiguous space, and it can impact system performance.

You can obtain the pointer to the chunk using FtrGet. To write to the chunk, you must use DmWrite because the chunk is in the storage heap, not the dynamic heap.

For example, if you allocate a memory chunk in this way:

```c
FtrPtrNew(appCreator,  
    myFtrMemPtr, 32, &ftrMem);
```

You can later access that memory and write to it using the following:

```c
void* data;  
if (!FtrGet(appCreator,  
    myFtrMemPtr, (UInt32*)&data))  
    DmWrite(data, 0, &someVal, sizeof(someVal));
```

**Compatibility**  
Implemented only if 3.1 New Feature Set is present.

**See Also**  
FtrPtrResize
**FtrPtrResize**

**Purpose**  
Resize feature memory.

**Declared In**  
FeatureMgr.h

**Prototype**  
Err FtrPtrResize (UInt32 creator, UInt16 featureNum, UInt32 newSize, void **newPtrP)

**Parameters**  
-> creator  
The creator ID for the feature.
-> featureNum  
Feature number of the feature.
-> newSize  
New size in bytes for the chunk.
<- newPtrP  
Pointer to the memory chunk is returned here.

**Result**  
Returns 0 if no error, or ftrErrNoSuchFeature if the specified feature number doesn’t exist for the specified creator, memErrInvalidParam if newSize is 0, or memErrNotEnoughSpace if there’s not enough free space available to allocate a chunk of that size.

**Comments**  
Use this function to resize a chunk of memory previously allocated by FtrPtrNew.

This function may move the chunk to a new location in order to resize it, so it is important to use the pointer returned by this function when accessing the memory chunk. The pointer in the feature table is automatically updated to be the same as the pointer returned by this function.

If this function fails, the old memory pointer still exists and its data is unchanged.

**Compatibility**  
Implemented only if 3.1 New Feature Set is present.

**See Also**  
MemHandleResize
**FtrSet**

**Purpose**
Set a feature.

**Declared In**
FeatureMgr.h

**Prototype**
Err FtrSet (UInt32 creator, UInt16 featureNum, UInt32 newValue)

**Parameters**
- `creator` Creator ID, which must be registered with PalmSource, Inc. This is usually the same as the creator ID for the application that owns this feature.
- `featureNum` Feature number for this feature.
- `newValue` New value.

**Result**
Returns 0 if no error, or memErrNotEnoughSpace if the feature table must be resized to add a new feature and no space is available.

**Comments**
The value of the feature is application-dependent.

A feature that you define in this manner remains defined until the next system reset or until you explicitly undefine the feature with FtrUnregister.

**See Also**
FtrGet, FtrPtrNew

**FtrUnregister**

**Purpose**
Unregister a feature.

**Declared In**
FeatureMgr.h

**Prototype**
Err FtrUnregister (UInt32 creator, UInt16 featureNum)

**Parameters**
- `creator` Creator ID for the feature.
-> featureNum  Feature number of the feature.

**Result**  Returns 0 if no error, or ftrErrNoSuchFeature if the specified feature number doesn’t exist for the specified creator.
File Streaming

This chapter provides reference material for the file streaming API.

- File Streaming Constants
- File Streaming Functions
- File Streaming Error Codes

The header file FileStream.h declares the API that this chapter describes. For more information on file streaming, see the chapter “Files and Databases” in the Palm OS Programmer’s Companion, vol. I.

File Streaming Constants

Primary Open Mode Constants

This section lists constants passed in the openMode parameter to the FileOpen function. These constants specify the mode in which a file stream is opened.

For each file stream, you must pass to the FileOpen function only one of the primary mode selectors listed.

Constant Values

- FileModeReadOnly: Open for read-only access
- FileModeReadWrite: Open/create for read/write access, discarding any previous version of stream
- FileModeUpdate: Open/create for read/write, preserving previous version of stream if it exists
- FileModeAppend: Open/create for read/write, always writing to the end of the stream
Secondary Open Mode Constants

You can use the | operator (bitwise inclusive OR) to append to a primary mode selector one or more of the secondary mode selectors listed below.

- **fileModeDontOverwrite**: Prevents fileModeReadWrite from discarding an existing stream having the same name; may only be specified together with fileModeReadWrite.

- **fileModeLeaveOpen**: Leave stream open when application quits. Most applications should not use this option.

- **fileModeExclusive**: No other application can open the stream until the application that opened it in this mode closes it.

- **fileModeAnyTypeCreator**: Accept any type/creator when opening or replacing an existing stream. Normally, the FileOpen function opens only streams having the specified creator and type. Setting this option enables the FileOpen function to open streams having a type or creator other than those specified.

- **fileModeTemporary**: Delete the stream automatically when it is closed. For more information, see Comment section of FileOpen function description.
File Streaming Functions

FileClearerr

Purpose Clear I/O error status, end of file error status, and last error.

Declared In FileStream.h

Prototype \texttt{Err FileClearerr (FileHand stream)}

Parameters \texttt{stream} Handle to open stream.

Result 0 if no error, or a fileErr code if an error occurs. See the section “File Streaming Error Codes” for more information.

Compatibility Implemented only if \texttt{3.0 New Feature Set} is present.

See Also FileGetLastError, FileRewind

FileClose

Purpose Close the file stream and destroy its handle. If the stream was opened with \texttt{fileModeTemporary}, it is deleted upon closing.

Declared In FileStream.h

Prototype \texttt{Err FileClose (FileHand stream)}

Parameters \texttt{stream} Handle to open stream.

Result 0 if no error, or a fileErr code if an error occurs. See the section “File Streaming Error Codes” for more information.

Compatibility Implemented only if \texttt{3.0 New Feature Set} is present.
File Streaming
File Streaming Functions

FileControl

Purpose
Perform the operation specified by the op parameter on the stream file stream.

Declared In
FileStream.h

Prototype
Err FileControl (FileOpEnum op, FileHand stream, void *valueP, Int32 *valueLenP)

Parameters

- **op**
  - The operation to perform, and its associated formal parameters. See the Comments section for a list of possible values.

- **stream**
  - Open stream handle if required for file stream operation.

- **valueP**
  - Pointer to value or buffer, as required. This parameter is defined by the selector passed as the value of the op parameter. For details, see the Comments section.

- **valueLenP**
  - Pointer to value or buffer, as required. This parameter is defined by the selector passed as the value of the op parameter. For details, see the Comments section.

Result
Returns either a value defined by the selector passed as the argument to the op parameter, or an error code resulting from the requested operation. For details, see the Comments section.

Comments
Normally, you do not call the FileControl function yourself; it is called for you by most of the other file streaming functions and macros to perform common file streaming operations. You can call FileControl yourself to enable specialized read modes.
fileOpNone

No-op.

fileOpDestructiveReadMode

Enter destructive read mode, and rewind stream to its beginning. Once in this mode, there is no turning back: stream’s contents after closing (or crash) are undefined.

Destructive read mode deletes blocks as data are read, thus freeing storage automatically. Once in destructive read mode, you cannot re-use the file stream—the contents of the stream are undefined after it is closed or after a crash.

Writing to files opened without write access or those that are in destructive read state is not allowed; thus, you cannot call the FileWrite, FileSeek, or FileTruncate functions on a stream that is in destructive read mode. One exception to this rule applies to streams that were opened in “write + append” mode and then switched into destructive read state. In this case, the FileWrite function can append data to the stream, but it also preserves the current stream position so that subsequent reads pick up where they left off (you can think of this as a pseudo-pipe).

ARGUMENTS:
stream = open stream handle
valueP = NULL
valueLenP = NULL

RETURNS:
zero on success;
fileErr... on error

fileOpGetEOFStatus

Get end-of-file status (like C runtime’s f eof) (err = fileErrEOF). Indicates end of file condition. Use FileClearerr to clear this error status.
ARGUMENTS:
stream = open stream handle
valueP = NULL
valueLenP = NULL

RETURNS:
zero if not end of file;
non-zero if end of file

fileOpGetLastError
Get error code from last operation on stream, and clear the last error code value. Doesn’t change status of EOF or I/O errors — use FileClearerr to reset all error codes.

ARGUMENTS:
stream = open stream handle
valueP = NULL
valueLenP = NULL

RETURNS:
Error code from last file stream operation

fileOpClearError
Clear I/O and EOF error status and last error.

ARGUMENTS:
stream = open stream handle
valueP = NULL
valueLenP = NULL

RETURNS:
zero on success; fileErr... on error

fileOpGetIOErrorStatus
Get I/O error status (like C runtime’s ferror). Use FileClearerr to clear this error status.

ARGUMENTS:
stream = open stream handle
valueP = NULL
valueLenP = NULL

RETURNS:
zero if not I/O error;
non-zero if I/O error is pending.
fileOpGetCreatedStatus Find out whether file was created by FileOpen function

ARGUMENTS:
stream = open stream handle
valueP = Pointer to Boolean
valueLenP = Pointer to Int32 variable set to sizeof(Boolean)

RETURNS:
zero on success; fileErr... on error. The Boolean variable will be set to non-zero if the file was created.

fileOpGetOpenDbRef Get the open database reference (handle) of the underlying database that implements the stream (NULL if none); this is needed for performing Palm OS-specific operations on the underlying database, such as changing or getting creator and type, version, backup/reset bits, and so on.

ARGUMENTS:
stream = open stream handle
valueP = Pointer to DmOpenRef variable
valueLenP = Pointer to Int32 variable set to sizeof(DmOpenRef)

RETURNS:
zero on success; fileErr... on error. The DmOpenRef variable will be set to the file’s open db reference that may be passed to Data Manager calls; WARNING: Do not make any changes to the data of the underlying database -- doing so will corrupt the file stream.

fileOpFlush Flush any cached data to storage.
ARGUMENTS:
stream = open stream handle
valueP = NULL
valueLenP = NULL

RETURNS:
zero on success; fileErr... on error;

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also FileClearerr, FileEOF, FileError, FileFlush, FileGetLastError, FileRewind

FileDelete

Purpose Deletes the specified file stream from the specified card. Only a closed stream may be passed to this function.

Declared In FileStream.h

Prototype Err FileDelete (UInt16 cardNo, const Char *nameP)

Parameters

cardNo Card on which the file stream to delete resides. Currently, no Palm OS® devices support multiple cards, so this value must be 0.

nameP String that is the name of the stream to delete.

Result 0 if no error, or a fileErr code if an error occurs. See the section “File Streaming Error Codes” for more information.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also FileOpen
**File Streaming**

*File Streaming Functions*

---

**FileDmRead**

**Purpose**  
A macro that reads data from a file stream into a chunk, record, or resource residing in a database.

**Declared In**  
FileStream.h

**Prototype**  
```c
Int32 FileDmRead (FileHand stream,  
void *startOfDmChunkP, Int32 destOffset,  
Int32 objSize, Int32 numObj, Err *errP)
```

**Parameters**

---> stream  
Handle to open stream.

---> startOfDmChunkP  
Pointer to beginning of chunk, record or resource residing in a database.

destOffset  
Offset from startOfDmChunkP (base pointer) to the destination area (must be >= 0).

objSize  
Size of each stream object to read.

numObj  
Number of stream objects to read.

<---> errP  
Pointer to variable that is to hold the error code returned by this function. Pass NULL to ignore. See the section “File Streaming Error Codes” for more information.

**Result**  
The number of whole objects that were read—note that the number of objects actually read may be less than the number requested.

**Comments**  
When the number of objects actually read is less than the number requested, you may be able to determine the cause of this result by examining the return value of the errP parameter or by calling the `FileGetLastError` function. If the cause is insufficient data in the stream to satisfy the full request, the current stream position is at end-of-file and the “end of file” indicator is set. If a non-NULL pointer was passed as the value of the errP parameter when the FileDmRead function was called and an error was encountered, *errP holds a non-zero error code when the function returns. In
addition, the FileError and FileEOF functions may be used to check for I/O errors.

**Compatibility**

Implemented only if 3.0 New Feature Set is present.

**See Also**

FileRead, FileError, FileEOF

---

**File EOF**

**Purpose**

Get end-of-file status (err = fileErrEOF indicates end of file condition).

**Declared In**

FileStream.h

**Prototype**

Err FileEOF (FileHand stream)

**Parameters**

--> stream Handle to open stream.

**Result**

0 if not end of file; non-zero if end of file. See the section “File Streaming Error Codes” for more information.

**Comments**

This function’s behavior is similar to that of the feof function provided by the C programming language runtime library. Use FileClearerr to clear the I/O error status.

**Compatibility**

Implemented only if 3.0 New Feature Set is present.

**See Also**

FileClearerr, FileGetLastError, FileRewind
**FileError**

**Purpose**: Get I/O error status.

**Declared In**: FileStream.h

**Prototype**: `Err FileError (FileHand stream)`

**Parameters**

-- » stream Handle to open stream.

**Result**: 0 if no error, and non-zero if an I/O error indicator has been set for this stream. See the section “File Streaming Error Codes” for more information.

**Comments**: This function’s behavior is similar to that of the C programming language’s ferror runtime function.

Use `FileClearerr` to clear the I/O error status.

**Compatibility**: Implemented only if 3.0 New Feature Set is present.

**See Also**: FileClearerr, FileGetLastError, FileRewind

---

**FileFlush**

**Purpose**: Flush cached data to storage.

**Declared In**: FileStream.h

**Prototype**: `Err FileFlush (FileHand stream)`

**Parameters**

-- » stream Handle to open stream.

**Result**: 0 if no error, or a fileErr code if an error occurs. See the section “File Streaming Error Codes” for more information.

**Comments**: It is not always necessary to call this function explicitly—certain operations flush the contents of a stream automatically; for example,
streams are flushed when they are closed. Because this function’s behavior is similar to that of the `fflush` function provided by the C programming language runtime library, you only need to call it explicitly under circumstances similar to those in which you would call `fflush` explicitly.

**Compatibility**

Implemented only if [3.0 New Feature Set](#) is present.

---

**FileGetLastError**

**Purpose**

Get error code from last operation on file stream, and clear the last error code value (will not change end of file or I/O error status -- use `FileClearerr` to reset all error codes)

**Declared In**

`FileStream.h`

**Prototype**

```c
Err FileGetLastError (FileHand stream)
```

**Parameters**

```c
--stream Handle to open stream.
```

**Result**

Error code returned by the last file stream operation. See the section “File Streaming Error Codes” for more information.

**Compatibility**

Implemented only if [3.0 New Feature Set](#) is present.

**See Also**

`FileClearerr`, `FileEOF`, `FileError`
FileOpen

Purpose Open existing file stream or create an open file stream for I/O in the mode specified by the openMode parameter.

Declared In FileStream.h

Prototype FileHand FileOpen (UInt16 cardNo, const Char *nameP, UInt32 type, UInt32 creator, UInt32 openMode, Err *errP)

Parameters

- cardNo Card on which the file stream to open resides. Currently, no Palm Powered™ devices support multiple cards, so this value must be 0.

- -> nameP Pointer to text string that is the name of the file stream to open or create. This value must be a valid name—no wildcards allowed, must not be NULL.

- type File type of stream to open or create. Pass 0 for wildcard, in which case sysFileTFStream is used if the stream needs to be created and FileModeTemporary is not specified. If type is 0 and FileModeTemporary is specified, then sysFileTTemp is used for the file type of the stream this function creates.

- creator Creator of stream to open or create. Pass 0 for wildcard, in which case the current application’s creator ID is used for the creator of the stream this function creates.

- openMode Mode in which to open the file stream. You must specify only one primary mode selector. Additionally, you can use the | operator (bitwise inclusive OR) to append one or more secondary mode selectors to the primary mode selector. See “Primary Open Mode Constants” and “Secondary Open Mode Constants” for the list of possible values.
File Streaming
File Streaming Functions

<--> errP   Pointer to variable that is to hold the error code returned by this function. Pass NULL to ignore. See the section “File Streaming Error Codes” for a list of error codes.

Result   If successful, returns a handle to an open file stream; otherwise, returns 0.

In some cases, on some platforms, FileOpen returns a non-zero value when it has failed to open a file; thus, it is always a good idea to check the errP parameter value to determine if an error has occurred.

Comments   The FileModeReadOnly, FileModeReadWrite, FileModeUpdate, and FileModeAppend modes are mutually exclusive—pass only one of them to the FileOpen function!

When the FileModeTemporary open mode is used and the file type passed to FileOpen is 0, the FileOpen function uses sysFileTTemp (defined in SystemMgr.rh) for the file type, as recommended. In future versions of Palm OS, this configuration will enable the automatic cleanup of undeleted temporary files after a system crash. Automatic post-crash cleanup is not implemented in current versions of Palm OS.

To open a file stream even if it has a different type and creator than specified, pass the FileModeAnyTypeCreator selector as a flag in the openMode parameter to the FileOpen function.

The FileModeLeaveOpen mode is an esoteric option that most applications should not use. It may be useful for a library that needs to open a stream from the current application’s context and keep it open even after the current application quits. By default, Palm OS automatically closes all databases that were opened in a particular application’s context when that application quits. The FileModeLeaveOpen option overrides this default behavior.

Compatibility   Implemented only if 3.0 New Feature Set is present.
FileRead

Purpose
A macro that reads data from a stream into a buffer. Do not use this macro to read data into a chunk, record or resource residing in a database—you must use the FileDmRead macro for such operations.

Declared In
FileStream.h

Prototype
Int32 FileRead (FileHand stream, void *bufP, Int32 objSize, Int32 numObj, Err *errP)

Parameters
--->  stream     Handle to open stream.
--->  bufP       Pointer to beginning of buffer into which data is read
objSize       Size of each stream object to read.
numObj        Number of stream objects to read.
<--->  errP     Pointer to variable that is to hold the error code returned by this function. Pass NULL to ignore. See the section “File Streaming Error Codes” for a list of error codes.

Result
The number of whole objects that were read—note that the number of objects actually read may be less than the number requested.

Comments
Do not use this macro to read data into a chunk, record or resource residing in a database—you must use the FileDmRead macro for such operations.

When the number of objects actually read is fewer than the number requested, you may be able to determine the cause of this result by examining the return value of the errP parameter or by calling the FileGetLastError function. If the cause is insufficient data in the stream to satisfy the full request, the current stream position is at end-of-file and the “end of file” indicator is set. If a non-NULL pointer was passed as the value of the errP parameter when the FileRead function was called and an error was encountered, *errP holds a non-zero error code when the function returns. In
addition, the FileError and FileEOF functions may be used to check for I/O errors.

**Compatibility**  Implemented only if 3.0 New Feature Set is present.

**See Also**  FileDmRead

### FileRewind

**Purpose**  Reset position marker to beginning of stream and clear all error codes.

**Declared In**  FileStream.h

**Prototype**  Err FileRewind (FileHand stream)

**Parameters**  
- `stream`  Handle to open stream.

**Result**  0 if no error, or a fileErr code if an error occurs. See the section “File Streaming Error Codes” for more information.

**Compatibility**  Implemented only if 3.0 New Feature Set is present.

**See Also**  FileSeek, FileTell, FileClearerr, FileEOF, FileError, FileGetLastError

### FileSeek

**Purpose**  Set current position within a file stream, extending the stream as necessary if it was opened with write access.

**Declared In**  FileStream.h

**Prototype**  Err FileSeek (FileHand stream, Int32 offset, FileOriginEnum origin)

**Parameters**  
- `stream`  Handle to open stream.
offset

Position to set, expressed as the number of bytes from origin. This value may be positive, negative, or 0.

origin

Describes the origin of the position change. Possible values are:

fileOriginBeginning

From the beginning (first data byte of file).

fileOriginCurrent

From the current position.

fileOriginEnd

From the end of file (one position beyond last data byte).

Result

0 if no error, or a fileErr code if an error occurs. See the section "File Streaming Error Codes" for more information.

Comments

Attempting to seek beyond end-of-file in a read-only stream results in an I/O error. This function’s behavior is similar to that of the fseek function provided by the C programming language runtime library.

Compatibility

Implemented only if 3.0 New Feature Set is present.

See Also

FileRewind, FileTell

FileTell

Purpose

Retrieves the current position and, optionally, file size, of a stream.

Declared In

FileStream.h

Prototype

Int32 FileTell (FileHand stream, Int32 *fileSizeP, Err *errP)

Parameters

-> stream Handle to open stream.
File Streaming

File Streaming Functions

<- fileSizeP	Pointer to variable that holds value describing size of stream in bytes when this function returns. Pass NULL to ignore.

<-- errP	Pointer to variable that is to hold the error code returned by this function. Pass NULL to ignore. See the section “File Streaming Error Codes” for a list of possible error codes.

Result	If successful, returns current position, expressed as an offset in bytes from the beginning of the stream. If an error was encountered, returns -1 as a signed long integer.

Comments	The FileTell function can return the size of the input stream; as such, it provides some of the functionality of the standard C library stat function. Note, however, that unlike the stat function, FileTell requires that the file be open.

Compatibility	Implemented only if 3.0 New Feature Set is present.

See Also	FileRewind, FileSeek

FileTruncate

Purpose	Truncate the file stream to a specified size; not allowed on streams open in destructive read mode or read-only mode.

Declared In	FileStream.h

Prototype	Err FileTruncate (FileHand stream, Int32 newSize)

Parameters	---> stream	Handle of open stream.
newSize	New size; must not exceed current stream size.

Result	0 if no error, or a fileErr code if an error occurs. See the section “File Streaming Error Codes” for a list of possible error codes.
Compatibility
Implemented only if 3.0 New Feature Set is present.

See Also
FileTell

FileWrite

Purpose
Write data to a stream.

Declared In
FileStream.h

Prototype
Int32 FileWrite (FileHand stream,
const void *dataP, Int32 objSize, Int32 numObj,
Err *errP)

Parameters
--> stream Handle to open stream.
--> dataP Pointer to buffer holding data to write.
objSize Size of each stream object to write; must be ≥ 0.
numObj Number of stream objects to write.
<-- errP Optional pointer to variable that holds the error code returned by this function. Pass NULL to ignore. See the section “File Streaming Error Codes” for a list of possible error codes.

Result
The number of whole objects that were written—note that the number of objects actually written may be less than the number requested. Should available storage be insufficient to satisfy the entire request, as much of the requested data as possible is written to the stream, which may result in the last object in the stream being incomplete.

Comments
Writing to files opened without write access or those that are in destructive read state is not allowed; thus, you cannot call the FileWrite, FileSeek, or FileTruncate functions on a stream that is in destructive read mode. One exception to this rule applies to streams that were opened in “write + append” mode and then switched into destructive read state. In this case, the FileWrite function can append data to the stream, but it also preserves the
current stream position so that subsequent reads pick up where they left off (you can think of this as a pseudo-pipe).

**Compatibility**  Implemented only if [3.0 New Feature Set](#) is present.
# File Streaming Error Codes

This section lists all error codes returned by the file streaming functions.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileErrMemErr</td>
<td>(fileErrorClass</td>
<td>1)</td>
</tr>
<tr>
<td>fileErrInvalidParam</td>
<td>(fileErrorClass</td>
<td>2)</td>
</tr>
<tr>
<td>fileErrCorruptFile</td>
<td>(fileErrorClass</td>
<td>3)</td>
</tr>
<tr>
<td>fileErrNotFound</td>
<td>(fileErrorClass</td>
<td>4)</td>
</tr>
<tr>
<td>fileErrTypeCreatorMismatch</td>
<td>(fileErrorClass</td>
<td>5)</td>
</tr>
<tr>
<td>fileErrReplaceError</td>
<td>(fileErrorClass</td>
<td>6)</td>
</tr>
<tr>
<td>fileErrCreateError</td>
<td>(fileErrorClass</td>
<td>7)</td>
</tr>
<tr>
<td>fileErrOpenError</td>
<td>(fileErrorClass</td>
<td>8)</td>
</tr>
<tr>
<td>fileErrInUse</td>
<td>(fileErrorClass</td>
<td>9)</td>
</tr>
<tr>
<td>fileErrReadOnly</td>
<td>(fileErrorClass</td>
<td>10)</td>
</tr>
<tr>
<td>fileErrInvalidDescriptor</td>
<td>(fileErrorClass</td>
<td>11)</td>
</tr>
</tbody>
</table>
**File Streaming**

*File Streaming Error Codes*

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileErrCloseError</td>
<td>(fileErrorClass</td>
<td>12)</td>
</tr>
<tr>
<td>fileErrOutOfBounds</td>
<td>(fileErrorClass</td>
<td>13)</td>
</tr>
<tr>
<td>fileErrPermissionDenied</td>
<td>(fileErrorClass</td>
<td>14)</td>
</tr>
<tr>
<td>fileErrIOError</td>
<td>(fileErrorClass</td>
<td>15)</td>
</tr>
<tr>
<td>fileErrEOF</td>
<td>(fileErrorClass</td>
<td>16)</td>
</tr>
<tr>
<td>fileErrNotStream</td>
<td>(fileErrorClass</td>
<td>17)</td>
</tr>
</tbody>
</table>
Float Manager

This chapter provides reference material for the Float Manager API as follows:

- Float Manager Data Structures
- Float Manager Functions

The Float Manager API is declared in the header file FloatMgr.h. For more information on the Float Manager, see the section “Floating-Point” in the Palm OS Programmer’s Companion, vol. I.

Float Manager Data Structures

FlpCompDouble

Float Manager functions accept and require values of type FlpDouble. The FlpCompDouble union allows you to declare values that can be interpreted either as a double or as an FlpDouble. As well, this union contains fields that provide easy access to the component parts of the double-precision floating-point number.

```c
typedef union {
    double d;
    FlpDouble fd;
    UInt32 ul[2];
    FlpDoubleBits fdb;
} FlpCompDouble
```
Field Descriptions

d Provides access to the value as a double.

fd Provides access to the value as a FlpDouble, which can be passed to or received from many Float Manager functions.

ul Provides access to the value as two long integers.

fdb Provides access to specific fields.

FlpDoubleBits

This structure provides direct access to the component parts of an IEEE-754 double-precision floating-point number. Use the FlpCompDouble union to convert numbers of type double to and from FlpDoubleBits.

typedef struct {
    UInt32 sign : 1;
    Int32 exp : 11;
    UInt32 manH : 20;
    UInt32 manL;
} FlpDoubleBits

Field Descriptions

sign The sign bit. You can also use the FlpGetSign macro to obtain the sign bit, and the FlpNegate, FlpSetNegative, and FlpSetPositive macros to set the sign bit.

exp The bits that make up the exponent. You can also use the FlpGetExponent macro to obtain the exponent value.

manH The most-significant 20 bits of the mantissa.

manL The least-significant 32 bits of the mantissa.
Float Manager Functions

FlpAToF

Purpose
Convert a null-terminated ASCII string to a 64-bit floating-point number. The string must have the format:

\[ [+|][-][\text{digits}][.][\text{digits}][e|E[+|-][\text{digits}] \]

Declared In
FloatMgr.h

Prototype
FlpDouble FlpAToF (const Char *s)

Parameters
-> s
Pointer to the string to be converted.

Result
Returns the value of the string as a floating-point number.

Comment
The mantissa of the number is limited to 32 bits.
This function is close to being compatible with the ISO C library function \texttt{atof}. \texttt{atof} requires the form:

\[ [+|][-]\text{digits}[.][\text{digits}][e|E[+|-][\text{digits}] \]

In order to maintain backward compatibility with the Float Manager in Palm OS 1.0 (which could be used up to, but not including, Palm OS 4.0), this function considers all of the “digits” sections to be optional. Here’s a table showing the ISO and Palm OS behavior with some sample strings:

<table>
<thead>
<tr>
<th>String</th>
<th>ISO</th>
<th>&gt;= Palm OS 4.0</th>
<th>&lt; Palm OS 4.0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>“+”</td>
<td>+0</td>
<td>+0</td>
<td>+0</td>
<td></td>
</tr>
<tr>
<td>“.3”</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>“0.3e123”</td>
<td>0.3e123</td>
<td>0.3e123</td>
<td>0.3e12</td>
<td>The old Float Manager only allowed a 1 or 2 digit exponent.</td>
</tr>
</tbody>
</table>
Float Manager
Float Manager Functions

<table>
<thead>
<tr>
<th>String</th>
<th>ISO</th>
<th>&gt;= Palm OS 4.0</th>
<th>&lt; Palm OS 4.0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>“+1”</td>
<td>1</td>
<td>1</td>
<td>+0</td>
<td>The old Float Manager doesn’t allow a leading ‘+’ sign.</td>
</tr>
<tr>
<td>“1e+2”</td>
<td>1e2</td>
<td>1e2</td>
<td>1</td>
<td>The old Float Manager doesn’t allow a ‘+’ sign in the exponent.</td>
</tr>
<tr>
<td>“0.3E3”</td>
<td>0.3e3</td>
<td>0.3e3</td>
<td>0.3</td>
<td>The old Float Manager doesn’t allow a capital 'E' to mark the exponent.</td>
</tr>
<tr>
<td>“4294967297”</td>
<td>4294967297</td>
<td>4294967297</td>
<td>1</td>
<td>The old Float Manager uses an unsigned long and wraps around.</td>
</tr>
</tbody>
</table>

1. Using the old Float Manager documented in Appendix C, “1.0 Float Manager,” on page 2355.

Unlike `atof`, `FlpAToF` doesn’t accept leading white-space characters and it doesn’t accept decimal point characters other than ‘.’.

Compatibility
Implemented only if 2.0 New Feature Set is present. GCC users must use `FlpBufferAToF` instead of this function.

See Also
`FlpFToA`

FlpBase10Info

Purpose
Extract detailed information on the base 10 form of a floating-point number: the base 10 mantissa, exponent, and sign.

Declared In
`FloatMgr.h`

Prototype
```
Err FlpBase10Info (FlpDouble a, UInt32 *mantissaP, Int16 *exponentP, Int16 *signP)
```

Parameters
- `a` The floating-point number.
<- mantissaP  The base 10 mantissa.
<- exponentP  The base 10 exponent.
<- signP      The sign: 1 if the number is negative, 0 otherwise.

Result Returns 0 if no error, or flpErrOutOfRange if the supplied floating-point number is either not a number (NaN) or is infinite.

Comments The mantissa is normalized so it contains at least 8 significant digits when printed as an integer value.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also FlpGetExponent, FlpGetSign

FlpBufferAToF

Purpose Convert a null-terminated ASCII string to a floating-point number. The string must be in the format: [-]x[.]yyyyyyyy[e[-]zz]

Declared In FloatMgr.h

Prototype void FlpBufferAToF (FlpDouble *result, const Char *s)

Parameters <- result Pointer to the structure into which the return value is placed.
-> s Pointer to the null-terminated ASCII string to be converted.

Result Returns the value of the string as a floating-point number.

Comments See FlpAToF for a complete description of this function.

Compatibility Implemented only if 2.0 New Feature Set is present. Because the Palm OS ABI was not well-specified in this area, GCC by default implemented structure return differently from the compiler used to
build the ROM. As a result, GCC users must use this function instead of FlpAToF. CodeWarrior users can use either function; they are binary compatible.

**FlpBufferCorrectedAdd**

**Purpose**
Adds two floating-point numbers and corrects for least-significant-bit errors when the result should be zero but is instead very close to zero.

**Declared In**
FloatMgr.h

**Prototype**
```c
void FlpBufferCorrectedAdd (FlpDouble *result, FlpDouble firstOperand, FlpDouble secondOperand, Int16 howAccurate)
```

**Parameters**
- `<- result`  Pointer to the structure into which the return value is placed.
- `-> firstOperand`  The first of the two numbers to be added.
- `-> secondOperand`  The second of the two numbers to be added.
- `-> howAccurate`  The smallest difference in exponents that won’t force the result to zero. The value returned from this function is forced to zero if the difference between exponents in the smaller of the two operands and the result exceeds this value. Supply a value of zero for this parameter to obtain the default level of accuracy (which is equivalent to a howAccurate value of 48).

**Result**
Returns the calculated result.

**Comments**
See FlpCorrectedAdd for a complete description of this function.

**Compatibility**
Implemented only if 2.0 New Feature Set is present. Because the Palm OS ABI was not well-specified in this area, GCC by default
implemented structure return differently from the compiler used to build the ROM. As a result, GCC users must use this function instead of `FlpCorrectedAdd`. CodeWarrior users can use either function; they are binary compatible.

**FlpBufferCorrectedSub**

**Purpose** Subtracts two floating-point numbers and corrects for least-significant-bit errors when the result should be zero but is instead very close to zero.

**Declared In** `FloatMgr.h`

**Prototype**

```c
void FlpBufferCorrectedSub (FlpDouble *result, FlpDouble firstOperand, FlpDouble secondOperand, Int16 howAccurate)
```

**Parameters**

- `<- result` Pointer to the structure into which the return value is placed.
- `-> firstOperand` The value from which `secondOperand` is to be subtracted.
- `-> secondOperand` The value to subtract from `firstOperand`.
- `-> howAccurate` The smallest difference in exponents that won’t force the result to zero. The value returned from this function is forced to zero if the difference between exponents in the smaller of the two operands and the result exceeds this value. Supply a value of zero for this parameter to obtain the default level of accuracy (which is equivalent to a `howAccurate` value of 48).

**Result** Returns the calculated result.

**Comments** See `FlpCorrectedSub` for a complete description of this function.
Float Manager
Float Manager Functions

Compatibility  Implemented only if 2.0 New Feature Set is present. Because the Palm OS ABI was not well-specified in this area, GCC by default implemented structure return differently from the compiler used to build the ROM. As a result, GCC users must use this function instead of FlpCorrectedSub. CodeWarrior users can use either function; they are binary compatible.

FlpCorrectedAdd

Purpose  Adds two floating-point numbers and corrects for least-significant-bit errors when the result should be zero but is instead very close to zero.

Declared In  FloatMgr.h

Prototype  FlpDouble FlpCorrectedAdd
            (FlpDouble firstOperand, FlpDouble secondOperand, Int16 howAccurate)

Parameters  -> firstOperand  The first of the two numbers to be added.
            -> secondOperand  The second of the two numbers to be added.
            -> howAccurate  The smallest difference in exponents that won’t force the result to zero. The value returned from FlpCorrectedAdd is forced to zero if, when the exponent of the result of the addition is subtracted from the exponent of the smaller of the two operands, the difference exceeds the value specified for howAccurate. Supply a value of zero for this parameter to obtain the default level of accuracy (which is equivalent to a howAccurate value of 48).

Result  Returns the calculated result.

Comments  Adding or subtracting a large number and a small number produces a result similar in magnitude to the larger number. Adding or
subtracting two numbers that are similar in magnitude can, depending on their signs, produce a result with a very small exponent (that is, a negative exponent that is large in magnitude). If the difference between the result’s exponent and that of the operands is close to the number of significant bits expressible by the mantissa, it is quite possible that the result should in fact be zero.

There also exist cases where it may be useful to retain accuracy in the low-order bits of the mantissa. For instance: 99999999 + 0.00000001 - 99999999. However, unless the fractional part is an exact (negative) power of two, it is doubtful that what few bits of mantissa that are available will be enough to properly represent the fractional value. In this example, the 99999999 requires 26 bits, leaving 26 bits for the .00000001; this guarantees inaccuracy after the subtraction.

The problem arises from the difficulty in representing decimal fractions such as 0.1 in binary. After about three successive additions or subtractions, errors begin to appear in the least significant bits of the mantissa. If the value represented by the most significant bits of the mantissa is then subtracted away, the least significant bit error is normalized and becomes the actual result—when in fact the result should be zero.

This problem is only an issue for addition and subtraction.

**Compatibility**

Implemented only if 2.0 New Feature Set is present. GCC users must use FlpBufferCorrectedAdd instead of this function.

**See Also**

FlpCorrectedSub
FlpCorrectedSub

**Purpose**
Subtracts two floating-point numbers and corrects for least-significant-bit errors when the result should be zero but is instead very close to zero.

**Declared In**
FloatMgr.h

**Prototype**
FlpDouble FlpCorrectedSub
(FlpDouble firstOperand, FlpDouble secondOperand, Int16 howAccurate)

**Parameters**
- `-> firstOperand` The value from which `secondOperand` is to be subtracted.
- `-> secondOperand` The value to subtract from `firstOperand`.
- `-> howAccurate` The smallest difference in exponents that won’t force the result to zero. The value returned from FlpCorrectedSub is forced to zero if, when the exponent of the result of the subtraction is subtracted from the exponent of the smaller of the two operands, the difference exceeds the value specified for `howAccurate`. Supply a value of zero for this parameter to obtain the default level of accuracy (which is equivalent to a `howAccurate` value of 48).

**Result**
Returns the calculated result.

**Comments**
See the comments for FlpCorrectedAdd.

**Compatibility**
Implemented only if 2.0 New Feature Set is present. GCC users must use FlpBufferCorrectedSub instead of this function.
FlpFToA

**Purpose**
Convert a floating-point number to a null-terminated ASCII string in exponential format: \([-x.yyyyyyyye[-]zz\]

**Declared In**
FloatMgr.h

**Prototype**
Err FlpFToA (FlpDouble a, Char *s)

**Parameters**
- `a`
  Floating-point number.
- `s`
  Pointer to buffer to contain the ASCII string.

**Result**
Returns 0 if no error, or `flpErrOutOfRange` if the supplied value is infinite or is not a number. In this case, the buffer is set to the string “INF”, “-INF”, or “NaN” as appropriate.

**Compatibility**
Implemented only if [2.0 New Feature Set](#) is present.

**See Also**
FlpAToF

FlpGetExponent

**Purpose**
Macro that returns the exponent of a 64-bit floating-point value. The returned value has the bias applied, so it ranges from -1023 to +1024.

**Declared In**
FloatMgr.h

**Prototype**
FlpGetExponent (x)

**Parameters**
- `x`
  The value from which the exponent is to be extracted.

**Result**
Returns a `UInt32` containing the exponent of the specified value.

**Compatibility**
Implemented only if [2.0 New Feature Set](#) is present.

**See Also**
FlpBase10Info, FlpGetSign
**Float Manager**

*Float Manager Functions*

---

**FlpGetSign**

**Purpose**  
Macro that returns the sign of a 64-bit floating-point value.

**Declared In**  
FloatMgr.h

**Prototype**  
`FlpGetSign (x)`

**Parameters**  
`-> x`  
The value from which the sign bit is to be extracted.

**Result**  
Returns a `UInt32` with a nonzero value if the specified value is negative, and with a zero value if it is positive.

**Compatibility**  
Implemented only if [2.0 New Feature Set](#) is present.

**See Also**  
`FlpBase10Info`, `FlpGetExponent`, `FlpNegate`, `FlpSetNegative`, `FlpSetPositive`

---

**FlpIsZero**

**Purpose**  
Macro that returns whether the specified 64-bit floating-point value is zero.

**Declared In**  
FloatMgr.h

**Prototype**  
`FlpIsZero (x)`

**Parameters**  
`-> x`  
The value for which the sign bit is desired.

**Result**  
Returns a `UInt32` with a nonzero value if the specified value is zero, and with a zero value if the specified value is other than zero.

**Compatibility**  
Implemented only if [2.0 New Feature Set](#) is present.
FlpNegate

**Purpose** Macro that changes the sign bit of a 64-bit floating-point number.

**Declared In** FloatMgr.h

**Prototype** FlpNegate (x)

**Parameters** -> x The value in which the sign bit is to be changed.

**Result** Returns a 64-bit floating-point value which is the negative of the value specified by x.

**Compatibility** Implemented only if 2.0 New Feature Set is present.

**See Also** FlpGetSign, FlpSetNegative, FlpSetPositive

FlpSetNegative

**Purpose** Macro that ensures that a 64-bit floating-point number is negative.

**Declared In** FloatMgr.h

**Prototype** FlpSetNegative (x)

**Parameters** -> x The value that is to be forced negative.

**Result** If the supplied 64-bit floating-point value is negative, that value is returned unchanged. If the supplied value is positive, the negative of that value is returned.

**Compatibility** Implemented only if 2.0 New Feature Set is present.

**See Also** FlpGetSign, FlpNegate, FlpSetPositive
**Float Manager**  
*Float Manager Functions*

**FlpSetPositive**

**Purpose**  
Macro that ensures that a 64-bit floating-point number is positive.

**Declared In**  
FloatMgr.h

**Prototype**  
FlpSetPositive (x)

**Parameters**
-> x  
The value that is to be forced positive.

**Result**  
If the supplied 64-bit floating-point value is positive, that value is returned unchanged. If the supplied value is negative, its absolute value is returned.

**Compatibility**  
Implemented only if 2.0 New Feature Set is present.

**See Also**  
FlpGetSign, FlpNegate, FlpSetNegative

**FlpVersion**

**Purpose**  
Returns the version number of the Float Manager.

**Declared In**  
FloatMgr.h

**Prototype**  
UInt32 FlpVersion (void)

**Parameters**  
None.

**Result**  
Returns the version number of the Float Manager. The current version is represented by the constant flpVersion, which is defined in FloatMgr.h.

**Compatibility**  
Implemented only if 2.0 New Feature Set is present.
Fonts

This chapter provides the following information regarding font support:

- **Font Data Structures**
- **Font Constants**
- **Font Resources**
- **Font Functions**

The header files `Font.h` and `FontSelect.h` declare the API that this chapter describes. For more information on fonts, see Chapter 8, “Text,” on page 251 of the *Palm OS Programmer’s Companion*, vol. I.

Font Data Structures

**FontCharInfoPtr**

The `FontCharInfoPtr` type points to a `FontCharInfoType` structure.

```
typedef FontCharInfoType *FontCharInfoPtr;
```

**FontCharInfoType**

The `FontCharInfoType` structure defines an entry in the offset/width table for a font.

**WARNING!** Palm, Inc. does not support or provide backward compatibility for the `FontCharInfoType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.
typedef struct FontCharInfoTag {
    Int8 offset;
    Int8 width;
} FontCharInfoType;

Field Descriptions

offset  This value is not currently used and must be set to 0.

width   The exact width in pixels of the glyph. You can retrieve this
        information using the function FntWCharWidth or
        FntCharWidth.

FontDensityType

The FontDensityType structure defines an entry in the
densities array in the FontTypeV2 structure. The densities
array specifies the location of each set of glyphs within an extended
font resource.

WARNING!  Palm, Inc. does not support or provide backward
compatibility for the FontDensityType structure. Never access
its structure members directly, or your code may break in future
versions. Use the information below for debugging purposes only.

typedef struct FontDensityTag {
    Int16 density;
    UInt32 glyphBitsOffset;
} FontDensityType;

Field Description

density      Either kDensityLow or kDensityDouble.
glyphBitsOffset  Offset in bytes from the
                 beginning of the font data to the
                 start of the font image for this
density.
Compatibility
This structure is only defined if the High-Density Display Feature Set is present.

FontID
The FontID enum specifies the IDs of available fonts. A font can either be a system-defined font or an application-defined font. You can obtain the ID of the current font using FntGetFont and change the font using FntSetFont.

enum fontID {
    stdFont = 0x00,
    boldFont,
    largeFont,
    symbolFont,
    symbol11Font,
    symbol7Font,
    ledFont,
    largeBoldFont,
    fntAppCustomBase = 0x80
};

typedef enum fontID FontID;

Value Descriptions

stdFont A small standard font used to display user input. This font is small to display as much text as possible.

boldFont Same size as stdFont but bold for easier reading. Used for text labels in the user interface.

largeFont A larger font provided as an alternative for users who find the standard font too small to read.

symbolFont Contains many special characters such as arrows, Graffiti® Shift Indicators, and so on.
Fonts
Font Data Structures

symbol11Font Contains the check boxes, the large left arrow, and the large right arrow.
symbol7Font Contains the up and down arrows used for the repeating button scroll arrows and the dimmed version of the same arrows.
ledFont Contains the numbers 0 through 9, –, ., and the comma (,). Used by the Calculator application for its numeric display.
largeBoldFont In Palm OS® 3.0 and later only. Same size as largeFont but bold.
fntAppCustomBase The first available ID for application-defined fonts.

FontPtr
The FontPtr type defines a pointer to a FontType structure.

typedef FontType *FontPtr;

FontType
The FontType structure defines a font resource’s header. The fields in this structure give general information about the font. Following the structure are several tables that Palm OS uses to draw the font on the screen. See “Font Resource” on page 718 for more information about the font resource.

WARNING! Palm, Inc. does not support or provide backward compatibility for the FontType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

typedef struct FontTag {
    Int16 fontType;
    Int16 firstChar;
    Int16 lastChar;
    Int16 maxWidth;
}
Font Data Structures

```c
Int16 kernMax;
Int16 nDescent;
Int16 fRectWidth;
Int16 fRectHeight;
Int16 owTLoc;
Int16 ascent;
Int16 descent;
Int16 leading;
Int16 rowWords;
} FontType;
```

**Field Descriptions**

- **fontType**
  A mask providing the general characteristics of the font. When creating an application-defined font resource, use 0x9000.

- **firstChar**
  Character code of first glyph in the font.

- **lastChar**
  Character code of last glyph in the font.

- **maxWidth**
  The maximum width in pixels of any glyph. In Palm OS, there is currently no difference between this field and `fRectWidth`.

- **kernMax**
  This value is not currently used and must be set to 0.

- **nDescent**
  This value is not currently used and must be set to 0.

- **fRectWidth**
  A metric of the font image. In Palm OS, this metric is equivalent to the maximum width in pixels of any glyph in the font. Use `FntAverageCharWidth` to obtain this value.

- **fRectHeight**
  The height, including ascenders and descenders, of the glyphs in this font. Use `FntCharHeight` to obtain this value.
### New FontTypeV2

The FontTypeV2 structure defines the header for an extended font resource, which contains a separate set of glyphs for each screen density. Currently the only supported densities are `kDensityLow` and `kDensityDouble`. See “Extended Font Resource” on page 721 for more information.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owTLoc</td>
<td>The offset in 16-bit words from this field to the first byte of the offset/width table. The offset/width table is a table of <code>FontCharInfoType</code> structures giving the width of each character in the font. Do not access the offset/width table directly. Use <code>FntWCharWidth</code> or <code>FntCharWidth</code> instead.</td>
</tr>
<tr>
<td>ascent</td>
<td>The distance in pixels from the top of the font rectangle to its baseline. Use <code>FntBaseLine</code> to obtain this value.</td>
</tr>
<tr>
<td>descent</td>
<td>The distance in pixels from the baseline to the bottom of the font rectangle. Use <code>FntDescenderHeight</code> to obtain this value.</td>
</tr>
<tr>
<td>leading</td>
<td>The font’s leading, which is the vertical space between lines of text, in pixels. This field is unused in Palm OS and must be set to 0. If your font requires a leading value, add blank space to the bottom of each of your glyphs. The <code>FntLineHeight</code> function returns the size of the font’s character cell plus the leading.</td>
</tr>
<tr>
<td>rowWords</td>
<td>The number of 16-bit words stored for each row of a glyph’s bitmap where <code>fRectHeight</code> is the number of rows.</td>
</tr>
</tbody>
</table>
WARNING! Palm, Inc. does not support or provide backward compatibility for the FontCharTypeV2 structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```c
typedef struct FontTagV2 {
    Int16 fontType;
    Int16 firstChar;
    Int16 lastChar;
    Int16 maxWidth;
    Int16 kernMax;
    Int16 nDescent;
    Int16 fRectWidth;
    Int16 fRectHeight;
    Int16 owTLoc;
    Int16 ascent;
    Int16 descent;
    Int16 leading;
    Int16 rowWords;
    Int16 version;
    Int16 densityCount;
    FontDensityType densities[0];
} FontTypeV2;
```

NOTE: All pixel values given in the fields below are in terms of the single-density font data.

Field Descriptions

- **fontType**: A mask providing the general characteristics of the font. When creating an application-defined extended font resource, use the value `fntExtendedFormatMask | 0x9000`.
- **firstChar**: Character code of first glyph in the font.
- **lastChar**: Character code of last glyph in the font.
### Fonts

#### Font Data Structures

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxWidth</td>
<td>The maximum width in pixels of any glyph. In Palm OS, there is currently no difference between this field and fRectWidth.</td>
</tr>
<tr>
<td>kernMax</td>
<td>This value is not currently used and must be set to 0.</td>
</tr>
<tr>
<td>nDescent</td>
<td>This value is not currently used and must be set to 0.</td>
</tr>
<tr>
<td>fRectWidth</td>
<td>A metric of the font image. In Palm OS, this metric is equivalent to the maximum width in pixels of any glyph in the font. Use FntAverageCharWidth to obtain this value.</td>
</tr>
<tr>
<td>fRectHeight</td>
<td>The height, including ascenders and descenders, of the glyphs in this font. Use FntCharHeight to obtain this value.</td>
</tr>
<tr>
<td>owTLoc</td>
<td>The offset in 16-bit words from this field to the first byte of the offset/width table. The offset/width table is a table of FontCharInfoType structures giving the width of each character in the font. Do not access the offset/width table directly. Use FntWCharWidth or FntCharWidth instead.</td>
</tr>
<tr>
<td>ascent</td>
<td>The distance in pixels from the top of the font rectangle to its baseline. Use FntBaseline to obtain this value.</td>
</tr>
<tr>
<td>descent</td>
<td>The distance in pixels from the baseline to the bottom of the font rectangle. Use FntDescenderHeight to obtain this value.</td>
</tr>
<tr>
<td>leading</td>
<td>The font’s leading, which is the vertical space between lines of text, in pixels. This field is unused in Palm OS and must be set to 0. If your font requires a leading value, add blank space to the bottom of each of your glyphs. The FntLineHeight function returns the size of the font’s character cell plus the leading.</td>
</tr>
</tbody>
</table>
Compatibility
This structure is only defined if the High-Density Display Feature Set is present.

Font Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkboxFont</td>
<td>symbol11Font</td>
<td>A convenience constant that points to the font containing the checkbox bitmap.</td>
</tr>
<tr>
<td>fntMissingChar</td>
<td>-1</td>
<td>The value used for a character that does not have a definition in the current font. The missing character symbol is usually an open rectangle.</td>
</tr>
<tr>
<td>fntExtendedFormatMask</td>
<td>0x0200</td>
<td>A constant used for the fontType field of a font to indicate that it is an extended font resource.</td>
</tr>
<tr>
<td>fntTabChrWidth</td>
<td>20</td>
<td>The width of the tab character in pixels.</td>
</tr>
</tbody>
</table>
Font Resources

Font Resource

The font resource ('NFNT') represents a version 1 single-density font. This resource is the same as the Macintosh 'NFNT' resource with some restrictions. It contains a header followed by several tables that provide information about each glyph in the font.

Figure 33.1 shows how the font resource is laid out in memory. Table 33.1 describes each table within the font resource.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FontType</td>
<td>Contains general information about the glyphs in the font. See FontType.</td>
</tr>
<tr>
<td>header</td>
<td></td>
</tr>
<tr>
<td>Font image</td>
<td>A raw bitmap image containing the packed character glyphs from left to right (see Figure 33.2 on page 720). This part of the resource tells Palm OS how to draw each character in the font. The height of the image is fRectHeight and the size is rowWords * 2 * fRectHeight.</td>
</tr>
<tr>
<td></td>
<td>Place glyphs sequentially in order of increasing character code. Leave at least a one-pixel wide vertical column of space to the right of each image so that there is space between characters when Palm OS draws text on the screen. If your font requires leading, leave horizontal space at the bottom of the characters as well. The font image must end with the glyph for the missing character symbol.</td>
</tr>
<tr>
<td>Bitmap location</td>
<td>A table of 16-bit words that specify the location of each glyph’s entry in the font image. The location is specified as the bit offset from the start of the image to the glyph in the first row of the font image. The last entry in the table contains the offset of the column after the last bitmap. (See Figure 33.2 on page 720.)</td>
</tr>
<tr>
<td>table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If you have skipped characters within an encoding, for each glyph that is missing, specify the same value for its location as the entry for the next glyph in the table.</td>
</tr>
</tbody>
</table>

Table 33.1 Font resource description
Table 33.1 Font resource description (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offset/width table</td>
<td>A table that specifies how wide each glyph in the font is. On Macintosh systems, this table also specifies how each glyph kerns. Palm OS does not support kerning, as the offset value is ignored. Each entry in the offset/width table is two bytes long. The first byte should be 0, and the second byte should contain the glyph width, which must be greater than or equal to 0. If the glyph at this index does not have a bitmap in the font image, the values should be –1 and –1.</td>
</tr>
</tbody>
</table>

Figure 33.2 shows an example of the font image for a font that defines glyphs for four characters (A, B, C, and the missing character symbol) and the portion of the bitmap location table that provides the offsets for these characters. The last entry in the bitmap location table is the offset to the column after the last bitmap, or 0x0014.

Figure 33.2 Font image and bitmap location table

![Font image and bitmap location table](image)

Bitmap location table values: 0x0000 0x0005 0x000A 0x000F 0x0014
Extended Font Resource

The extended font resource (‘nfnt’) defines a font that supports multiple screen densities. Currently, only two screen densities are supported: the standard density of 80 dpi, as occurs on most devices that use a 160 X 160 pixel display, and double density of 160 dpi, as occurs on most devices that use a 320 X 320 pixel display. As shown in Figure 33.3, the extended font resource is essentially:

- A **FontTypeV2** header giving all general information about the glyphs in the font. All metrics are in terms of the low-density version of the font.
- Tables for the low-density font. See “Font Resource” on page 718 for a description of these tables.
- The font image (set of glyphs) for each density specified by the font.

Compatibility

This resource is only defined if the High-Density Display Feature Set is present.
Figure 33.3 Extended font resource
Font Functions

**FntAverageCharWidth**

*Purpose*  
Gets the *maximum* character width in pixels of the current font.

*Declared In*  
Font.h

*Prototype*  
Int16 FntAverageCharWidth (void)

*Parameters*  
None.

*Result*  
Returns the *maximum* character width (in pixels).

*Comments*  
This function returns the value of the `fRectHeight` field in the `FontType` structure for the current font. Because Palm OS does not support kerning, this value is the maximum width in pixels rather than the average width.

**FntBaseLine**

*Purpose*  
Gets the distance from the top of the character cell to the baseline for the current font.

*Declared In*  
Font.h

*Prototype*  
Int16 FntBaseLine (void)

*Parameters*  
None.

*Result*  
Returns the ascent of the font (in pixels).
Fonts
Font Functions

**FntCharHeight**

**Purpose**
Gets the character height of the current font including accents and descenders.

**Declared In**
Font.h

**Prototype**
Int16 FntCharHeight (void)

**Parameters**
None.

**Result**
Returns the height of the characters in the current font, expressed in pixels.

**FntCharsInWidth**

**Purpose**
Finds the length in bytes of the characters from a specified string that fit within a passed width.

**Declared In**
Font.h

**Prototype**
void FntCharsInWidth (Char const *string, Int16 *stringWidthP, Int16 *stringLengthP, Boolean *fitWithinWidth)

**Parameters**
- > string          A pointer to the character string.
  <-> stringWidthP  The maximum width to allow (in pixels). Upon return, contains the actual width allowed. Note that this value does not include any trailing spaces or tabs, which are stripped by this function.
<-> stringLengthP

The maximum length of text to allow, in bytes (assumes current font). Upon return, contains the number of bytes of text that can appear within the width. Note that this value does not include any trailing space or tabs, which are stripped by this function.

<-> fitWithinWidth

Upon return, false if the string is considered truncated, true if it isn’t.

**Result**

Returns nothing.

**Comments**

Spaces and tabs at the end of a string are ignored and removed. If the string fits within the specified width after spaces and tabs are removed, the fitWithinWidth value contains true. Characters after a carriage return are ignored, and the string is considered truncated.

This function is specifically designed for the code used to draw text fields. Consider using FntWidthToOffset in your application code instead, particularly if you do not want the special processing of trailing spaces, tabs, and carriage returns.

**FntCharsWidth**

**Purpose**

Gets the width of the specified character string. The missing character symbol (an open rectangle) is substituted for any character that does not exist in the current font.

**Declared In**

Font.h

**Prototype**

Int16 FntCharsWidth (Char const *chars, Int16 len)

**Parameters**

-> chars Pointer to a string of characters.
Font Functions

- `len` Length in bytes of the string.

Result
Returns the width of the string, in pixels.

Comments
Like all functions that work with strings, this function returns correct results for strings with multi-byte characters as well as strings with only single-byte characters.

See Also
FntCharWidth

FntCharWidth

Purpose
Gets the width of the specified character. If the specified character does not exist within the current font, the missing character symbol is substituted.

Declared In
Font.h

Prototype
`Int16 FntCharWidth (Char ch)`

Parameters
- `ch` Character whose width is needed.

Result
Returns the width of the specified character (in pixels).

Comments
FntCharWidth works with single-byte characters only. To determine the pixel width of a single-byte character or a multi-byte character, use FntWCharWidth instead of this function in Palm OS 4.0 and higher. Alternatively, for compatibility with earlier versions of Palm OS, link with the PalmOSGlue library and call FntGlueWCharWidth. For more information, see Chapter 75, "PalmOSGlue Library."

See Also
FntCharsWidth
**FntDefineFont**

**Purpose**
Makes a custom font available to your application.

**Declared In**
Font.h

**Prototype**
Err FntDefineFont (FontID font, FontPtr fontP)

**Parameters**
- `-> font` A value greater than or equal to fntAppFontCustomBase that identifies the custom font to the system. Values less than that are reserved for system use. Note that font IDs are 8-bit unsigned values and so must be less than 256. See FontID.

- `-> fontP` Pointer to the custom font resource to be used by this function. This resource must remain locked until the calling application undefines the custom font or quits.

**Result**
errNone No error
memErrNotEnoughSpace Insufficient dynamic heap space

**Comments**
The custom font is available only when the application that called this function is running; when the application quits, the custom font is uninstalled automatically.

The font this function specifies is not available at build time; as a result, some UI elements—labels, for example—cannot determine their bounds automatically as they do when using the built-in fonts.

Before you use this function, you must load the font resource from the database and obtain a pointer to it. See “Creating Custom Fonts” on page 275 of the Palm OS Programmer’s Companion, vol. I for more information.

**Compatibility**
Implemented only if 3.0 New Feature Set is present.

Palm OS 4.0 and later do not allow a NULL value for the fontP parameter; therefore, you cannot undefine a font using
FntDefineFont (myFontID, NULL). Earlier versions of Palm OS allowed the NULL value. Note that there is usually no need to undefine a font.

Palm OS 3.0 and 3.1 have problems with font ID values greater than 131. These problems are fixed in later releases of Palm OS.

See Also  FontSelect, FntSetFont

FntDescenderHeight

Purpose  Gets the height of a character’s descender in the current font. The height of a descender is the distance between the baseline and the bottom of the character cell.

Declared In  Font.h

Prototype  Int16 FntDescenderHeight (void)

Parameters  None.

Result  Returns the height of a descender, expressed in pixels.

FntGetFont

Purpose  Gets the font ID of the current font.

Declared In  Font.h

Prototype  FontID FntGetFont (void)

Parameters  None.

Result  Returns the ID of the current font.

Comments  The current font is the font stored in the draw state. It is used when drawing characters directly onto the screen. Most user interface
elements, such as fields, tables, labels, and buttons, do not use the current font.

See Also  FntSetFont, FntGetFontPtr, FontID

FntGetFontPtr

Purpose  Gets a pointer to the current font.

Declared In  Font.h

Prototype  FontPtr FntGetFontPtr (void)

Parameters  None.

Result  Returns a pointer to the current font.

Comments  The current font is the font stored in the draw state. It is used when drawing characters directly onto the screen. Most user interface elements, such as fields, tables, labels, and buttons, do not use the current font.

See Also  FntGetFont

FntGetScrollValues

Purpose  Gets the values needed to update a scroll bar based on a specified string and the position within the string.

Declared In  Font.h

Prototype  void FntGetScrollValues (Char const *chars, UInt16 width, UInt16 scrollPos, UInt16 *linesP, UInt16 *topLine)

Parameters  -> chars  A null-terminated string.
### width

The width of a line of text in the display, given in pixels.

### scrollPos

The byte offset of the first character displayed on the topmost line.

### linesP

Number of lines required to display the string.

### topLine

The line of text that is the topmost visible line. Line numbering starts with 0.

**Result**

Returns nothing. Stores the number of lines of text in `linesP` and the top visible line in `topLine`.

**Compatibility**

Implemented only if [2.0 New Feature Set](#) is present.

**See Also**

[FldGetScrollValues](#)

### FntIsAppDefined

**Purpose**

Macro that returns `true` if the font is defined by the application or `false` if it is defined by the system.

**Declared In**

Font.h

**Prototype**

`FntIsAppDefined (fnt)`

**Parameters**

- `fnt`
  
  The `FontID` of a font.

**Result**

Boolean that indicates if the font is an application-defined font. Returns `true` if application-defined, `false` if system-defined.
FntLineHeight

**Purpose**
Gets the height of a line in the current font. The height of a line is the height of the character cell plus the space between lines (the external leading).

**Declared In**
Font.h

**Prototype**
Int16 FntLineHeight (void)

**Parameters**
None.

**Result**
Returns the height in pixels of a line in the current font.

FntLineWidth

**Purpose**
Gets the width of the specified line of text, taking tab characters into account. The function assumes that the characters passed are left-aligned and that the first character in the string is the first character drawn on a line. In other words, this routine doesn’t work for characters that don’t start at the beginning of a line.

**Declared In**
Font.h

**Prototype**
Int16 FntLineWidth (Char const *pChars, UInt16 length)

**Parameters**
- `pChars` Pointer to a string of characters.
- `length` Length in bytes of the string.

**Result**
Returns the line width (in pixels).
FntSetFont

Purpose
Sets the current font.

Declared In
Font.h

Prototype
FontID FntSetFont (FontID font)

Parameters
-> font ID of the font to make the current font.

Result
Returns the ID of the previous font.

Comments
If the specified font ID is invalid, this function sets the current font to stdFont.

The current font is the font stored in the draw state. It is used when drawing characters directly onto the screen. Most user interface elements, such as fields, tables, labels, and buttons, do not use the current font. To set the font for one of these elements, check the API for that element. If the element’s API doesn’t have a function to set the font programmatically, check the PalmOSGlue library.

See Also
FntGetFont

FntWCharWidth

Purpose
Gets the width of the specified character. If the specified character does not exist within the current font, the missing character symbol is substituted.

Declared In
Font.h

Prototype
Int16 FntWCharWidth (WChar iChar)

Parameters
-> iChar Character whose width is needed.

Result
Returns the width of the specified character (in pixels).
Comments   FntWCharWidth works with both single-byte characters and multi-byte characters. However, you should always pass a WChar variable to this function rather than a Char to avoid sign extension problems on values 0x80 and higher.

Compatibility   Implemented only if 4.0 New Feature Set is present. If you want to use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call FntGlueWCharWidth. For more information, see Chapter 75, “PalmOSGlue Library.”

See Also   FntCharWidth

FntWidthToOffset

Purpose   Given a pixel position, gets the offset of the character displayed at that location.

Declared In   Font.h

Prototype   Int16 FntWidthToOffset (Char const *pChars, Uint16 length, Int16 pixelWidth, Boolean *leadingEdge, Int16 *truncWidth)

Parameters   -> pChars   Pointer to the character string.
   -> length   The length in bytes of pChars.
   -> pixelWidth   A horizontal pixel offset from the beginning of the string.
   <- leadingEdge   Set to true if the pixel position pixelWidth falls on the left side of the character. Pass NULL for this parameter if you don’t need this information.
<-truncWidth  The width of the text (in pixels) up to but not including the returned offset. Pass NULL for this parameter if you don’t need this information.

**Result**  Returns the byte offset into pChars of the character that contains the pixel offset pixelWidth. If pixelWidth is past the right edge of the string, the function returns the byte offset past the last character in pChars, and truncWidth contains the width required to display the entire string.

**Compatibility**  Implemented only if 3.1 New Feature Set is present. If you want to use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call FntGlueWidthToOffset. For more information, see Chapter 75, “PalmOSGlue Library.”

### FntWordWrap

**Purpose**  Given a string, determines how many bytes of text can be displayed within the specified width with a line break at a tab or space character.

**Declared In**  Font.h

**Prototype**  UInt16 FntWordWrap (Char const *chars, UInt16 maxWidth)

**Parameters**  
- `chars`  A pointer to a null-terminated string.
- `maxWidth`  The maximum line width in pixels.

**Result**  Returns the length of the line, in bytes. If the entire string cannot be displayed within maxWidth, the value that this function returns specifies the offset where the line should be broken, which is typically following a space, tab, or line-feed character.

**Compatibility**  Implemented only if 2.0 New Feature Set is present.

**See Also**  FldWordWrap
**FntWordWrapReverseNLines**

**Purpose**  Word wraps a text string backwards by the number of lines specified. The character position of the start of the first line and the number of lines that are actually word wrapped are returned.

**Declared In**  Font.h

**Prototype**  
```
void FntWordWrapReverseNLines (Char const *const chars, UInt16 maxWidth, 
UInt16 *linesToScrollP, UInt16 *scrollPosP)
```

**Parameters**  
- `chars`  A pointer to a null-terminated string.
- `maxWidth`  The maximum line width in pixels.
- `linesToScrollP`  The number of lines to scroll. Upon return, contains the number of lines that were scrolled.
- `scrollPosP`  The byte offset of the first character displayed on the topmost line. Upon return, contains the first character after wrapping.

**Result**  Returns nothing. Stores the first character after wrapping and the number of lines scrolled in `scrollPosP` and `linesToScrollP`.

**Compatibility**  Implemented only if 2.0 New Feature Set is present.
**FontSelect**

**Purpose**  Displays a dialog from which the user can choose one of the system-supplied fonts, and returns a `FontID` value representing the user’s choice.

**Declared In**  FontSelect.h

**Prototype**  

```c
FontID FontSelect (FontID fontID)
```

**Parameters**  

`fontID`  

A `FontID` value specifying the font to be highlighted as the default choice in the dialog box that this function displays. This value must be one of the following system-supplied constants:

- `stdFont`  
  Standard plain text font.

- `boldFont`  
  Bold version of `stdFont`.

- `largeFont`  
  A large plain text font (Japanese devices only).

- `largeBoldFont`  
  Larger version of `boldFont`.

**Result**  Returns a `FontID` value representing the font that the user chose.

**Comments**  When your application launches for the first time, it should determine the system’s default font. The default font varies based on locale. You can use `FntGlueGetDefaultFontID` from the PalmOSGlue library to determine the default font as follows:

```c
fntID = FntGlueGetDefaultFontID(defaultSystemFont);
```

**Compatibility**  Implemented only if 3.0 New Feature Set is present.

**See Also**  [FntGetFont](#), [FntSetFont](#)
This chapter provides reference material for the Graffiti® manager. The Graffiti manager API is declared in the header file Graffiti.h.

For more information on the Graffiti manager, see “The Graffiti Manager” on page 60 of the Palm OS Programmer’s Companion, vol. I.

**Graffiti Manager Functions**

**GrfAddMacro**

**Purpose**
Add a macro to the macro list.

**Declared In**
Graffiti.h

**Prototype**
Err GrfAddMacro (const Char *nameP, Uint8 *macroDataP, Uint16 dataLen)

**Parameters**
- nameP: Name of macro.
- macroDataP: Data of macro.
- dataLen: Size of macro data in bytes.

**Result**
Returns 0 if no error; returns grfErrNoMacros, grfErrMacroPtrTooSmall, dmErrNotValidRecord, dmErrWriteOutOfBounds if an error occurs.

**See Also**
GrfGetMacro, GrfGetMacroName, GrfDeleteMacro
**Graffiti Manager**

**Graffiti Manager Functions**

---

### GrfAddPoint

**Purpose**
Add a point to the Graffiti point buffer.

**Declared In**
Graffiti.h

**Prototype**
Err GrfAddPoint (PointType *pt)

**Parameters**
pt 
Pointer to point buffer.

**Result**
Returns 0 if no error; returns grfErrPointBufferFull if an error occurs.

**See Also**
GrfFlushPoints

### GrfCleanState

**Purpose**
Remove any temporary shifts from the dictionary state.

**Declared In**
Graffiti.h

**Prototype**
Err GrfCleanState (void)

**Parameters**
None

**Result**
Returns 0 if no error, or grfErrNoDictionary if an error occurs.

**See Also**
GrfInitState
GrfDeleteMacro

**Purpose**  Delete a macro from the macro list.

**Declared In**  Graffiti.h

**Prototype**  
```
Err GrfDeleteMacro (UInt16 index)
```

**Parameters**  

- **index**  Index of the macro to delete.

**Result**  Returns 0 if no error, or grfErrNoMacros, grfErrMacroNotFound if an error occurs.

**See Also**  GrfAddMacro

GrfFilterPoints

**Purpose**  Filter the points in the Graffiti point buffer.

**Declared In**  Graffiti.h

**Prototype**  
```
Err GrfFilterPoints (void)
```

**Parameters**  None.

**Result**  Always returns 0.

**See Also**  GrfMatch
GrfFindBranch

Purpose
Locate a branch in the Graffti dictionary by flags.

Declared In
Graffiti.h

Prototype
Err GrfFindBranch (UInt16 flags)

Parameters
flags
Flags of the branch you're searching for.

Result
Returns 0 if no error, or grfErrNoDictionary or grfErrBranchNotFound if an error occurs.

See Also
GrfCleanState, GrfInitState

GrfFlushPoints

Purpose
Dispose of all points in the Graffti point buffer.

Declared In
Graffiti.h

Prototype
Err GrfFlushPoints (void)

Parameters
None.

Result
Always returns 0.

See Also
GrfAddPoint
GrfGetAndExpandMacro

**Purpose**
Look up and expand a macro in the current macros.

**Declared In**
Graffiti.h

**Prototype**
```
Err GrfGetAndExpandMacro (Char *nameP, 
UInt8 *macroDataP, UInt16 *dataLenP)
```

**Parameters**
- nameP: Name of macro to look up.
- macroDataP: Macro contents returned here.
- dataLenP: On entry, size of macroDataP buffer; on exit, number of bytes in macro data.

**Result**
Returns 0 if no error, or grfErrNoMacros or grfErrMacroNotFound if an error occurs.

**See Also**
GrfAddMacro, GrfGetMacro

GrfGetGlyphMapping

**Purpose**
Look up a glyph in the dictionary and return the text.

**Declared In**
Graffiti.h

**Prototype**
```
Err GrfGetGlyphMapping (UInt16 glyphID, 
UInt16 *flagsP, void *dataPtrP, UInt16 *dataLenP, 
UInt16 *uncertainLenP)
```

**Parameters**
- glyphID: Glyph ID to look up.
- flagsP: Returned dictionary flags.
- dataPtrP: Where returned text goes.
- dataLenP: On entry, size of dataPtrP; on exit, number of bytes returned.
uncertainLenP  Return number of uncertain characters in text.

**Result**  
Returns 0 if no error, or grfErrNoDictionary or grfErrNoMapping if an error occurs.

**See Also**  [GrfMatch](#)

**GrfGetMacro**

**Purpose**  Look up a macro in the current macros.

**Declared In**  Graffiti.h

**Prototype**  
Err GrfGetMacro (Char *nameP, UInt8 *macroDataP, UInt16 *dataLenP)

**Parameters**  
- nameP  Name of macro to lookup.
- macroDataP  Macro contents returned here.
- dataLenP  On entry: size of macroDataP buffer. On exit: number of bytes in macro data.

**Result**  
Returns 0 if no error or grfErrNoMacros, grfErrMacroNotFound.

**See Also**  [GrfAddMacro](#)

**GrfGetMacroName**

**Purpose**  Look up a macro name by index.

**Declared In**  Graffiti.h

**Prototype**  
Err GrfGetMacroName (UInt16 index, Char *nameP)

**Parameters**  
- index  Index of macro.
nameP Name returned here.

**Result** Returns 0 if no error, or grfErrNoMacros or grfErrMacroNotFound if an error occurs.

**See Also** GrfAddMacro, GrfGetMacro

### GrfGetNumPoints

**Purpose** Return the number of points in the point buffer.

**Declared In** Graffiti.h

**Prototype** Err GrfGetNumPoints (UInt16 *numPtsP)

**Parameters**
- numPtsP Returned number of points.

**Result** Always returns 0.

**See Also** GrfAddPoint

### GrfGetPoint

**Purpose** Return a point out of the Graffiti point buffer.

**Declared In** Graffiti.h

**Prototype** Err GrfGetPoint (UInt16 index, PointType *pointP)

**Parameters**
- index Index of the point to get.
- pointP Returned point.

**Result** Returns 0 if no error, or grfErrBadParam if an error occurs.

**See Also** GrfAddPoint, GrfGetNumPoints
**Grafitti Manager**

**Grafitti Manager Functions**

---

**GrfGetState**

**Purpose**

Return the current Grafitti shift state.

**Declared In**

Graffiti.h

**Prototype**

Err GrfGetState (Boolean *capsLockP,
Boolean *numLockP, UInt16 *tempShiftP,
Boolean *autoShiftedP)

**Parameters**

- capsLockP  
  Returns true if caps lock on.
- numLockP  
  Returns true if num lock on.
- tempShiftP  
  Current temporary shift.
- autoShiftedP  
  Returns TRUE if shift not set by the user but by the system, for example, at the beginning of a line.

**Result**

Always returns 0.

**Compatibility**

Palm OS® 2.0 and later has more user-friendly auto shifting. It uses an upper case letter under these conditions:

- after an empty field
- after a period or other sentence terminator (such as ? or !).
- after two spaces

**See Also**

GrfSetState
**GrfInitState**

**Purpose**
Reinitialize the Graffiti dictionary state.

**Declared In**
Graffiti.h

**Prototype**
Err GrfInitState (void)

**Parameters**
None.

**Result**
Always returns 0.

**See Also**
GrfGetState, GrfSetState

**GrfMatch**

**Purpose**
Recognize the current stroke in the Graffiti point buffer and return with the recognized text.

**Declared In**
Graffiti.h

**Prototype**
Err GrfMatch (UInt16 *flagsP, void *dataPtrP, UInt16 *dataLenP, UInt16 *uncertainLenP, GrfMatchInfoPtr matchInfoP)

**Parameters**
- flagsP: Glyph flags are returned here.
- dataPtrP: Return text is placed here.
- dataLenP: Size of dataPtrP on exit; number of characters returned on exit.
- uncertainLenP: Return number of uncertain characters.
- matchInfoP: Array of grfMaxMatches, or NULL.

**Result**
Returns 0 if no error, or grfErrNoGlyphTable, grfErrNoDictionary, or grfErrNoMapping if an error occurs.

**See Also**
GrfAddPoint, GrfFlushPoints
**GrfMatchGlyph**

**Purpose**
Recognize the current stroke as a glyph.

**Declared In**
Graffiti.h

**Prototype**
```
Err GrfMatchGlyph (GrfMatchInfoPtr matchInfoP,
Int16 maxUnCertainty, UInt16 maxMatches)
```

**Parameters**
- `matchInfoP` Pointer to array of matches to fill in.
- `maxUnCertainty` Maximum number of errors to tolerate.
- `maxMatches` Size of `matchInfoP` array.

**Result**
Returns 0 if no error, or `grfErrNoGlyphTable` if an error occurs.

**See Also**
GrfMatch

**GrfProcessStroke**

**Purpose**
Translate a stroke to keyboard events using Graffiti.

**Declared In**
Graffiti.h

**Prototype**
```
Err GrfProcessStroke (PointType *startPtP,
PointType *endPtP, Boolean upShift)
```
**Parameters**

- **startPtP**
  Start point of stroke.

- **endPtP**
  End point of stroke.

- **upShift**
  Set to `true` to feed an artificial upshift into the engine.

**Result**

Returns 0 if recognized.

**Comments**

Called by `SysHandleEvent` when a `penUpEvent` is detected in the writing area. This routine recognizes the stroke and sends the recognized characters into the key queue. It also flushes the stroke out of the pen queue after recognition.

**See Also**

- `SysHandleEvent`

**GrfSetState**

**Purpose**

Set the current shift state of Graffiti.

**Declared In**

`Graffiti.h`

**Prototype**

```c
Err GrfSetState ( Boolean capsLock, Boolean numLock, Boolean upperShift )
```

**Parameters**

- **capsLock**
  Set to `true` to turn oncaps lock.

- **numLock**
  Set to `true` to turn on num lock.

- **upperShift**
  Set to `true` to put into upper shift.

**Result**

Always returns 0.

**See Also**

- `GrfGetState`
Helper API

This chapter describes the Helper API declared in the header files Helper.h and HelperServiceClass.h. The Helper API is used when an application broadcasts a `sysNotifyHelperEvent` to all interested parties. The broadcaster of the notification and the notification clients (called helpers) use the Helper APIs to communicate with each other. The chapter discusses the following topics:

- Helper Data Structures
- Helper Constants

For more information on using the Helper API, see the section “Helper Notifications” on page 38 of the Palm OS Programmer’s Companion, vol. I.

Helper Data Structures

**HelperNotifyEnumerateListType**

The `HelperNotifyEnumerateListType` provides the broadcaster of the helper notification with information about the services that the helper can provide. This structure is used as the data field of the `HelperNotifyEventType` structure when the action code is `kHelperNotifyActionCodeEnumerate`.

```c
typedef struct HelperNotifyEnumerateListTypeTag
{
    struct HelperNotifyEnumerateListTypeTag *nextP;
    Char helperAppName[kHelperAppMaxNameSize];
    Char actionName[kHelperAppMaxActionNameSize];
    UInt32 helperAppID;
    UInt32 serviceClassID;
} HelperNotifyEnumerateListType;
```
Note that the helper allocates this structure and then adds it to the linked list of structures pointed to by notifyDetailsP->data.enumerateP in the SysNotifyParamType that is sent to the helper. The helper should allocate one structure per supported service.

Even though the helper allocates this structure, the helper is not responsible for freeing the structure. Instead, the application that broadcast the notification must free the structure.

Field Descriptions

nextP A pointer to the next element in the list or NULL to signal the end of the list.

helperAppName A null-terminated string containing the name of the helper application, suitable for display in the user interface. If more than one application can perform the same service, this name is displayed as one of the choices in a pop-up list.

actionName A null-terminated string containing the name of the service that can be performed, suitable for display in the user interface. The action name should be short enough to display on a button.

helperAppID The helper’s creator ID or any other ID that uniquely identifies the helper.

serviceClassID The ID of the service that the helper performs. See Helper Service Class IDs.

Compatibility Implemented only if 4.0 New Feature Set is present.
**HelperNotifyEventType**

The `HelperNotifyEventType` structure contains all data associated with a helper notification (sysNotifyHelperEvent). A pointer to this structure is passed as the notifyDetailsP field in the `SysNotifyParamType` for that notification.

```c
typedef struct HelperNotifyEventTypeTag {
    UInt16 version;
    HelperNotifyActionCodeType actionCode;
    union {
        struct HelperNotifyEnumerateListTypeTag
            *enumerateP;
        struct HelperNotifyValidateTypeTag
            *validateP;
        struct HelperNotifyExecuteTypeTag
            *executeP;
    } data;
} HelperNotifyEventType;
```

**Field Descriptions**

- **version**: The version number for this structure. The current version is 1.
- **actionCode**: The action that the helper application should perform. See Table 35.1.
- **data**: Data specific to the action code. See Table 35.1.

The `HelperNotifyEventType` structure specifies which action is to be performed and contains data necessary for that action. All actions have some common data. Actions also have data specific to that action. The specific data uses a union that is part of the `HelperNotifyEventType` structure.
Helper API
Helper Data Structures

Table 35.1 Helper action codes

<table>
<thead>
<tr>
<th>Action Code</th>
<th>data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kHelperNotifyActionCode Enumerate</td>
<td>HelperNotifyEnumerateListType</td>
<td>Send a list of available services.</td>
</tr>
<tr>
<td>kHelperNotifyActionCode Validate</td>
<td>HelperNotifyValidateType</td>
<td>Validate the input data for the specified service.</td>
</tr>
<tr>
<td>kHelperNotifyActionCode Execute</td>
<td>HelperNotifyExecuteType</td>
<td>Perform the specified service.</td>
</tr>
</tbody>
</table>

Compatibility
Implemented only if 4.0 New Feature Set is present.

HelperNotifyExecuteType

The HelperNotifyExecuteType structure identifies the service to perform and contains the data necessary to perform that service. This structure is used as the data field of the HelperNotifyEventType structure when the action code is kHelperNotifyActionCodeExecute.

typedef struct HelperNotifyExecuteTypeTag {
    UInt32  serviceClassID;
    UInt32  helperAppID;
    Char *dataP;
    Char *displayName;
    void *detailsP;
    Err err;
} HelperNotifyExecuteType;
Field Descriptions

serviceClassID  The ID of the service to be performed. See Helper Service Class IDs.

helperAppID  The unique ID of the helper; a value of 0 indicates that any available helper for the specified service class should perform the service.

dataP  A null-terminated string specific to this service, such as a phone number for the dial service or an email address for the email service. See Table 35.2. Multiple fields must be separated by semicolons (;).

displayedName  A null-terminated string containing an optional, human-readable description of the string in dataP. For example, if dataP contains a phone number, this field might contain the name of the person at that number.

detailsP  A pointer to a data structure containing extra information that this service requires. See Table 35.2. If the service does not require extra information, this field is NULL.

t  An error code that indicates if the service was performed successfully or not. If the service was successful, this field contains errNone, and the handled field in the notification data structure should be set to true.

The following table lists the Palm OS-defined values for the service class ID and for each service, shows what value dataP contains and what type of structure detailsP points to.
**Helper API**

**Helper Data Structures**

Table 35.2 HelperNotifyExecuteType values

<table>
<thead>
<tr>
<th>Service Class ID</th>
<th>dataP Value</th>
<th>detailsP Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>kHelper ServiceClassID</td>
<td>The telephone number to dial</td>
<td>NULL</td>
</tr>
<tr>
<td>VoiceDial</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kHelper ServiceClassID</td>
<td>The email address to which the</td>
<td>HelperServiceEMailDetailsType</td>
</tr>
<tr>
<td>EMail</td>
<td>message is to be sent</td>
<td></td>
</tr>
<tr>
<td>kHelper ServiceClassID</td>
<td>The SMS mailbox number to which</td>
<td>HelperServiceSMSDetailsType</td>
</tr>
<tr>
<td>SMS</td>
<td>the message is to be sent</td>
<td></td>
</tr>
<tr>
<td>kHelper ServiceClassID</td>
<td>The fax number to which the fax</td>
<td>NULL</td>
</tr>
<tr>
<td>Fax</td>
<td>is to be sent</td>
<td></td>
</tr>
</tbody>
</table>

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**HelperNotifyValidateType**

The HelperNotifyValidateType structure identifies a service that should be validated and the helper that should validate it. This structure is used as the data field of the HelperNotifyEventType structure when the action code is kHelperNotifyActionCodeValidate.

```c
typedef struct HelperNotifyValidateTypeTag {
    UInt32 serviceClassID;
    UInt32 helperAppID;
} HelperNotifyValidateType;
```
Field Descriptions

serviceClassID  The ID of the service to be validated. See Helper Service Class IDs.

helperAppID     The creator ID of the helper application. 0 indicates that any available helper for the specified service should respond. If nonzero, only the helper with the matching creator ID should respond.

The helper returns true in the handled field of the SysNotifyParamType structure to indicate that the service can be performed or false to indicate that the service cannot be performed.

Compatibility  Implemented only if 4.0 New Feature Set is present.

HelperServiceEMailDetailsType

The HelperServiceEMailDetailsType structure provides additional data for the email service. It is used as the detailsP field in the HelperNotifyExecuteType when the service class ID is kHelperServiceClassIDEMail.

typedef struct _HelperServiceEMailDetailsType

{ Uint16 version;
  Char *cc;
  Char *subject;
  Char *message;
} HelperServiceEMailDetailsType;

Field Descriptions

version  The version number for this structure. The current version is 1.

cc  A null-terminated string containing an email address that should be sent a carbon copy of the message. Multiple addresses are separated by a semi-colon (;). May be NULL if there are no email addresses to carbon copy.
Helper Service Class IDs

The header file HelperServiceClass.h defines the constants listed in Table 35.3 as service class IDs. Third party developers may define their own service classes. To do so, you must register a 32-bit identifier with PalmSource, Inc. on this web site:

http://www.palmos.com/dev/creatorid/

Alternatively, you can use a creator ID that you already own.

Compatibility

Implemented only if 4.0 New Feature Set is present.

Helper Constants

subject A null-terminated string containing the subject line. May be NULL.
message Initial message body string or NULL.

Field Descriptions

version The version number for this structure. The current version is 1.
message A null-terminated string containing the body of the message to be sent, or NULL.

Compatibility

Implemented only if 4.0 New Feature Set is present.
### Table 35.3 Service class ID constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kHelperServiceClassIDVoiceDial</td>
<td>'voic'</td>
<td>Dial a phone number for a voice telephone call.</td>
</tr>
<tr>
<td>kHelperServiceClassIDEMail</td>
<td>'mail'</td>
<td>Send an email message.</td>
</tr>
<tr>
<td>kHelperServiceClassIDSMS</td>
<td>'sms_'</td>
<td>Send an SMS message.</td>
</tr>
<tr>
<td>kHelperServiceClassIDFax</td>
<td>'fax_'</td>
<td>Send a fax.</td>
</tr>
</tbody>
</table>

**Compatibility**  Implemented only if 4.0 New Feature Set is present.
Helper API
Helper Constants
Key Manager

This chapter provides reference material for the key manager. The key manager API is declared in the header file KeyMgr.h.

For more information on the key manager, see “The Key Manager” on page 62 of the Palm OS Programmer’s Companion, vol. I.

Key Manager Functions

**KeyCurrentState**

**Purpose**
Return bit field with bits set for each key that is currently depressed.

**Declared In**
KeyMgr.h

**Prototype**
UInt32 KeyCurrentState (void)

**Parameters**
None.

**Result**
Returns a UInt32 with bits set for keys that are depressed. See keyBitPower, keyBitPageUp, keyBitPageDown, etc., in KeyMgr.h.

**Comments**
Called by applications that need to poll the keys.

**See Also**
KeyRates
**KeyRates**

**Purpose**  Get or set the key repeat rates.

**Declared In**  KeyMgr.h

**Prototype**  Err KeyRates (Boolean set, UInt16* initDelayP, UInt16* periodP, UInt16* doubleTapDelayP, Boolean* queueAheadP)

**Parameters**

- **set**  If true, settings are changed; if false, current settings are returned.

- **initDelayP**  Initial delay in ticks for a auto-repeat event.

- **periodP**  Auto-repeat rate specified as period in ticks.

- **doubleTapDelayP**  Maximum double-tap delay, in ticks.

- **queueAheadP**  If true, auto-repeating keeps queueing up key events if the queue has keys in it. If false, auto-repeat doesn’t enqueue keys unless the queue is already empty.

**Result**  Returns 0 if no error.

**See Also**  KeyCurrentState
KeystMask

**Purpose**  Specify which keys generate `keyDownEvents`.
You can specify this either by using this function or by using the `poweredOnKeyMask` modifier.

**Declared In**  KeyMgr.h

**Prototype**  `UInt32 KeystMask (UInt32 keyMask)`

**Parameters**  
- `keyMask`  Mask with bits set for those keys to generate `keyDownEvents` for.

**Result**  Returns the old `key Mask`.

**Compatibility**  Implemented only if 2.0 New Feature Set is present.
Locale Manager

This chapter describes the Locale Manager API as described in the header files LocaleMgr.h, Localize.h, and PalmLocale.h. It discusses the following topics:

- **Locale Manager Data Types**
- **Locale Manager Constants**
- **Locale Manager Functions**

For more information on the Locale Manager, see the chapter “Localized Applications” on page 363 of the *Palm OS Programmer’s Companion*, vol. I.

**Locale Manager Data Types**

**CountryType**

The CountryType defines a country code. The *Country Constants* in PalmLocale.h define the possible values for CountryType variables.

```c
typedef UInt8 CountryType;
```

**Compatibility**

Prior to version 4.0, CountryType was an enum in Preferences.h that defined only 33 country codes. The Palm OS® 4.0 definition of CountryType is compatible with the previous definition.

**LanguageType**

The LanguageType defines a language code. The *Language Constants* in PalmLocale.h define the possible values for LanguageType variables.
typedef UInt8 LanguageType;

**Compatibility**

The `LanguageType` definition was added in Palm OS 3.5. Prior to version 4.0, `LanguageType` was an enum in `Preferences.h` that defined only eight language codes. The Palm OS 4.0 definition of `LanguageType` is compatible with the previous definition.

**LmLocaleType**

The `LmLocaleType` struct defines the country and language used in a locale.

```c
struct _LmLocaleType {
    UInt16 language;
    UInt16 country;
};
typedef struct _LmLocaleType LmLocaleType;
```

**Field Descriptions**

- **language** One of the [Language Constants](#). This value identifies the language spoken in the current locale.
- **country** One of the [Country Constants](#). This value identifies the locale’s country, which helps to identify the language dialect. For example, a language of `English` specifies a different dialect if the country is `UnitedKingdom` than if it is `UnitedStates`.

Note that the `language` and `country` fields are type `UInt16` instead of `LanguageType` and `CountryType`.

**Compatibility**

The `LmLocaleType` is defined only if [4.0 New Feature Set](#) is present. It supersedes the `OmLocaleType` that was introduced with Palm OS 3.5. `LmLocaleType` is bit-compatible with `OmLocaleType`.

**NumberFormatType**

The `NumberFormatType` enum specifies how numbers are formatted. You can pass a `NumberFormatType` value to
LocGetNumberSeparators and receive the appropriate separator characters for thousands and decimals.

```c
typedef enum {
    nfCommaPeriod,
    nfPeriodComma,
    nfSpaceComma,
    nfApostrophePeriod,
    nfApostropheComma
} NumberFormatType;
```

**Value Descriptions**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nfCommaPeriod</td>
<td>Uses a comma (,) as the thousands separator and a period (.) as the decimal separator.</td>
</tr>
<tr>
<td>nfPeriodComma</td>
<td>Uses a period as the thousands separator and a comma as the decimal separator.</td>
</tr>
<tr>
<td>nfSpaceComma</td>
<td>Uses a space ( ) as the thousands separator and a comma as the decimal separator.</td>
</tr>
<tr>
<td>nfApostrophePeriod</td>
<td>Uses an apostrophe (’) as the thousands separator and a period as the decimal separator.</td>
</tr>
<tr>
<td>nfApostropheComma</td>
<td>Uses an apostrophe as the thousands separator and a comma as the decimal separator.</td>
</tr>
</tbody>
</table>

**Locale Manager Constants**

**Character Encoding Constants**

The PalmLocale.h file defines several character encoding constants that are used as values of CharEncodingType variables. The character encoding constants generally follow the format:

`charEncodingName`

where *Name* is the name of the character encoding.
The following table shows examples of the character encoding constants. For a complete list, see the `PalmLocale.h` file.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>charEncodingUnknown</td>
<td>Unknown to this version of Palm OS®</td>
</tr>
<tr>
<td>charEncodingAscii</td>
<td>ISO 646-1991</td>
</tr>
<tr>
<td>charEncodingISO8859_1</td>
<td>ISO 8859 Part 1 (also known as ISO Latin 1). This encoding is commonly used for the Roman alphabet</td>
</tr>
<tr>
<td>charEncodingPalmLatin</td>
<td>Palm OS version of Microsoft Windows code page 1252. This encoding is identical to code page 1252 with Palm-specific characters added in the control range.</td>
</tr>
<tr>
<td>charEncodingShiftJIS</td>
<td>Encoding for 0208-1990 with single-byte Japanese Katakana. This encoding is commonly used for Japanese alphabets.</td>
</tr>
<tr>
<td>charEncodingPalmSJIS</td>
<td>Palm OS version of Microsoft Windows code page 932. This encoding is identical to code page 932, with Palm-specific characters added in the control range and with a Yen symbol instead of the Reverse Solidus at location 0x5c.</td>
</tr>
<tr>
<td>charEncodingCP1252</td>
<td>Microsoft Windows extensions to ISO 8859 Part 1</td>
</tr>
<tr>
<td>charEncodingCP932</td>
<td>Microsoft Windows extensions to Shift JIS</td>
</tr>
<tr>
<td>charEncodingUTF8</td>
<td>Eight-bit safe encoding for Unicode</td>
</tr>
</tbody>
</table>
Country Constants
The PalmLocale.h file defines several country constants that are used as values of CountryType variables. The country type constants have the following format:

$cCountryName$

where $CountryName$ is the name of the country. There is one constant for each country identified in the ISO 3166 standard, which currently defines 239 countries.

The following table shows examples of the country type constants. For a complete list, see the PalmLocale.h file.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cAustralia</td>
<td>Australia</td>
</tr>
<tr>
<td>cAustria</td>
<td>Austria</td>
</tr>
<tr>
<td>cBelgium</td>
<td>Belgium</td>
</tr>
</tbody>
</table>

Language Constants
The PalmLocale.h file defines several language constants that are used as values of LanguageType variables. The language type constants have the following format:

$lLanguageName$

where $LanguageName$ is the name of the language. There is one constant for each language specified in the ISO 639 standard, which currently defines 137 languages.

The following table shows examples of the language type constants. For a complete list, see the PalmLocale.h file.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lEnglish</td>
<td>English</td>
</tr>
<tr>
<td>lFrench</td>
<td>French</td>
</tr>
<tr>
<td>lGerman</td>
<td>German</td>
</tr>
</tbody>
</table>
Locale Manager
Locale Manager Functions

Locale Manager Size Constants

You can use the Locale Manager size constants to determine the size of strings to allocate for some of the locale settings.

NOTE: The variables in the table below do not count the terminating null character. Therefore, you need to allocate a string of size \( k_{\text{MaxCountryNameLen}} + 1 \) to hold a country name, for example.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kMaxCountryNameLen</td>
<td>19</td>
<td>The maximum length of a country name string.</td>
</tr>
<tr>
<td>kMaxCurrencyNameLen</td>
<td>19</td>
<td>The maximum length of a currency name string.</td>
</tr>
<tr>
<td>kMaxCurrencySymbolLen</td>
<td>5</td>
<td>The maximum length of a currency symbol string.</td>
</tr>
</tbody>
</table>

Locale Manager Functions

LmGetLocaleSetting

Purpose
Return the requested setting for a given locale.

Declared In
LocaleMgr.h

Prototype
Err LmGetLocaleSetting (UInt16 iLocaleIndex, LmLocaleSettingChoice iChoice, void *oValue, UInt16 iValueSize)

Parameters
- \( \rightarrow \) iLocaleIndex Index of the locale whose settings you want to retrieve.
- \( \rightarrow \) iChoice The setting you want to retrieve. This is a constant in the form \( \text{lmChoiceSettingName} \). See Table 37.1 for a list of possible values.
<- oValue  The value of the iChoice setting. The size of this buffer depends on the value of iChoice, as shown in Table 37.1.

-> iValueSize  The size of the oValue buffer.

**Result**  Returns one of the following values:

errNone  Success.

lmErrBadLocaleIndex  iLocaleIndex is out of range.

lmErrSettingDataOverflow  The oValue buffer is too small to hold the setting’s value.

lmErrBadLocaleSettingChoice  The iChoice parameter contains an unknown or unsupported value.

**Comments**  This function accesses the private locale system resource and returns the requested information in the oValue parameter. The size and type of the oValue parameter depend on which setting you want to retrieve. Table 37.1 lists and describes the possible settings and the type of data returned in oValue for each setting. For fixed-size values, make sure that oValue is no larger than the returned value.

This function returns the default settings for the locale. Users can override many of the locale settings using the Preferences application. Applications should always honor the user’s preferences rather than the locale defaults. For this reason, it’s recommended that if a corresponding system preference is available, you should check it instead (using PrefGetPreference). Use LmGetLocaleSetting only if you want to retrieve values that the user cannot override (such as the country name or currency symbol) or if you want to retrieve information about a locale other than the current locale.
### Table 37.1 LmGetLocaleSetting choices and sizes

<table>
<thead>
<tr>
<th>lmChoice...</th>
<th>oValue Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CountryName</td>
<td>String buffer of size kMaxCountryNameLen+1 bytes</td>
<td>The name of the locale’s country.</td>
</tr>
<tr>
<td>CurrencyName</td>
<td>String buffer of size kMaxCurrencyNameLen+1 bytes</td>
<td>The name of the currency used in this locale.</td>
</tr>
<tr>
<td>CurrencySymbol</td>
<td>String buffer of size kMaxCurrencySymbolLen+1 bytes</td>
<td>The symbol used to denote monetary values in this locale.</td>
</tr>
<tr>
<td>CurrencyDecimalPlaces</td>
<td>UInt16</td>
<td>The number of decimal places that monetary values are typically given.</td>
</tr>
<tr>
<td>DateFormat</td>
<td>DateFormatType</td>
<td>The short date format used in this locale. For example: 95/12/31</td>
</tr>
<tr>
<td>Locale</td>
<td>LmLocaleType</td>
<td>A structure containing the locale’s language and country codes.</td>
</tr>
<tr>
<td>LongDateFormat</td>
<td>DateFormatType</td>
<td>The long date format used in this locale. For example: 31 Dec 1995</td>
</tr>
<tr>
<td>MeasurementSystem</td>
<td>MeasurementSystemType</td>
<td>The measurement system (metric system or English system) used in this locale.</td>
</tr>
<tr>
<td>NumberFormat</td>
<td>NumberFormatType</td>
<td>The format used for numbers, with regards to the thousands separator and the decimal point, in this locale.</td>
</tr>
</tbody>
</table>
### Table 37.1 LmGetLocaleSetting choices and sizes *(continued)*

<table>
<thead>
<tr>
<th>ImChoice...</th>
<th>oValue Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeFormat</td>
<td><code>TimeFormatType</code></td>
<td>The format used for time values in this locale.</td>
</tr>
<tr>
<td>TimeZone</td>
<td><code>Int16</code></td>
<td>The locale’s default time zone given as the number of minutes east of GMT.</td>
</tr>
<tr>
<td>UniqueCurrencySymbol</td>
<td>String buffer of size <code>kMaxCurrencySymbolLen+1</code> bytes</td>
<td>A unique symbol for monetary values.</td>
</tr>
<tr>
<td>WeekStartDay</td>
<td><code>UInt16</code></td>
<td>The first day of the week (Sunday or Monday) in this locale. Days of the week are numbered from 0 to 6 starting with Sunday = 0.</td>
</tr>
</tbody>
</table>

**Compatibility**

Implemented only if [4.0 New Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call LmGlueGetLocaleSetting. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also**

LmGetNumLocales, LmLocaleToIndex
**LmGetNumLocales**

**Purpose**
Return the number of known locales.

**Declared In**
LocaleMgr.h

**Prototype**
UInt16 LmGetNumLocales (void)

**Parameters**
None.

**Result**
Returns the number of locales that the locale system resource defines.

**Comments**
Use this function to obtain the range of possible values that you can pass as an index to `LmGetLocaleSetting`. If `LmGetNumLocales` returns 3, then `LmGetLocaleSetting` accepts indexes in the range of 0 to 2.

This function returns only the number of locales for which the ROM has locale information. It does not return the number of locales that could possibly be defined. For example, the system resource currently contains no locale whose language is `lHebrew` and country is `cIsrael`, even though that is a valid locale.

**Compatibility**
Implemented only if [4.0 New Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `LmGlueGetNumLocales`. For more information, see [Chapter 75](#), "PalmOSGlue Library."
**LmLocaleToIndex**

**Purpose**
Convert an `LmLocaleType` to an index.

**Declared In**
`LocaleMgr.h`

**Prototype**
```c
Err LmLocaleToIndex (const LmLocaleType *iLocale, UInt16 *oLocaleIndex)
```

**Parameters**
- `<iLocale>` The locale to convert.
- `<oLocaleIndex>` The index of `iLocale` upon return.

**Result**
Returns `errNone` upon success or `lmErrUnknownLocale` if the locale could not be found.

**Comments**
You can use this function to obtain a valid index to pass to `LmGetLocaleSetting`. For example, you might use the Overlay Manager routine `OmGetSystemLocale` to return the locale used on the current system and then pass that locale to this function to obtain its index.

```c
LmLocaleType locale;
Char oValue[kMaxCurrencySymbolLen+1];
UInt16 index;

OmGetSystemLocale(&locale);
LmLocaleToIndex(&locale, &index);
LmGetLocaleSetting(index,
    lmChoiceCurrencySymbol, oValue,
    sizeof(oValue));
```

The `LmLocaleType` that is passed in `iLocale` can use the constants `lmAnyCountry` or `lmAnyLanguage` as wildcards. For example, if the country is `lmAnyCountry`, `LmLocaleToIndex` returns the index of the first locale that matches the language.

**Compatibility**
Implemented only if [4.0 New Feature Set](http://www.sonyericsson.com/us/en/support/palmosapi) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call
Locale Manager
Locale Manager Functions

LmGlueLocaleToIndex. For more information, see Chapter 75, “PalmOSGlue Library.”
Memory Manager

This chapter provides reference information for the memory manager. The memory manager API is declared in the header file MemoryMgr.h.

For more information on the memory manager, see the chapter “Memory” in the Palm OS Programmer’s Companion, vol. I.

Memory Manager Functions

**MemCardInfo**

**Purpose**
Return information about a memory card.

**Declared In**
MemoryMgr.h

**Prototype**
Err MemCardInfo (UInt16 cardNo, Char* cardNameP, Char* manufNameP, UInt16* versionP, UInt32* crDateP, UInt32* romSizeP, UInt32* ramSizeP, UInt32* freeBytesP)

**Parameters**
cardNo Card number.
cardNameP Pointer to character array (32 bytes), or 0.
manufNameP Pointer to character array (32 bytes), or 0.
versionP Pointer to version variable, or 0.
crDateP Pointer to creation date variable, or 0.
romSizeP Pointer to ROM size variable, or 0.
ramSizeP Pointer to RAM size variable, or 0.
Memory Manager
Memory Manager Functions

freeBytesP Pointer to free byte-count variable, or 0.

Result Returns 0 if no error.

Comments Pass 0 for those variables that you don’t want returned.

MemCmp

Purpose Compare two blocks of memory.

NOTE: Blocks are compared as unsigned bytes.

Declared In MemoryMgr.h

Prototype Int16 MemCmp (const void* s1, const void* s2, Int32 numBytes)

Parameters s1, s2 Pointers to block of memory.
numBytes Number of bytes to compare.

Result Zero if they match, non-zero if not:
+ if s1 > s2
- if s1 < s2

Compatibility Implemented only if 2.0 New Feature Set is present.
MemCmp can be used to test the equality of blocks in memory on all versions that support MemCmp; however, testing the sort ordering of blocks in memory works reliably only on Palm OS® versions 3.5 and higher. On versions earlier than 3.2, MemCmp always returns a positive value if the blocks are unequal. On versions 3.2 and 3.3, MemCmp reliably returns positive to indicate s1 > s2 and negative to indicate s1 < s2 only if the characters that differ are less than 128 apart. If the difference is greater than that, MemCmp may return positive when it should return negative and vice versa.
MemDebugMode

Purpose  Return the current debugging mode of the memory manager.

Declared In  MemoryMgr.h

Prototype  UInt16 MemDebugMode (void)

Parameters  No parameters.

Result  Returns debug flags as described for MemSetDebugMode.

MemHandleCardNo

Purpose  Return the card number a chunk resides in.

Declared In  MemoryMgr.h

Prototype  UInt16 MemHandleCardNo (MemHandle h)

Parameters  -> h  Chunk handle.

Result  Returns the card number.

Comments  Call this routine to retrieve the card number (0 or 1) a movable chunk resides on.

See Also  MemPtrCardNo
MemHandleDataStorage

**Purpose**
Return `true` if the given handle is part of a data storage heap. If not, it’s a handle in the dynamic heap.

**Declared In**
MemoryMgr.h

**Prototype**
```c
Boolean MemHandleDataStorage (MemHandle h)
```

**Parameters**
- `h` Chunk handle.

**Result**
Returns `true` if the handle is part of a data storage heap.

**Comments**
Called by Fields package routines to determine if they need to worry about data storage write-protection when editing a text field.

**See Also**
MemPtrDataStorage

MemHandleFree

**Purpose**
Dispose of a movable chunk.

**Declared In**
MemoryMgr.h

**Prototype**
```c
Err MemHandleFree (MemHandle h)
```

**Parameters**
- `h` Chunk handle.

**Result**
Returns 0 if no error, or `memErrInvalidParam` if an error occurs.

**Comments**
Call this routine to dispose of a movable chunk.

**See Also**
MemHandleNew
MemHandleHeapID

**Purpose**  Return the heap ID of a chunk.

**Declared In**  MemoryMgr.h

**Prototype**  UInt16 MemHandleHeapID (MemHandle h)

**Parameters**  -> h  Chunk handle.

**Result**  Returns the heap ID of a chunk.

**Comments**  Call this routine to get the heap ID of the heap a chunk resides in.

**See Also**  MemPtrHeapID

MemHandleLock

**Purpose**  Lock a chunk and obtain a pointer to the chunk’s data.

**Declared In**  MemoryMgr.h

**Prototype**  MemPtr MemHandleLock (MemHandle h)

**Parameters**  -> h  Chunk handle.

**Result**  Returns a pointer to the chunk.

**Comments**  Call this routine to lock a chunk and obtain a pointer to the chunk. MemHandleLock and MemHandleUnlock should be used in pairs.

**See Also**  MemHandleNew, MemHandleUnlock
MemHandleNew

**Purpose**
Allocate a new movable chunk in the dynamic heap and returns a handle to it.

**Declared In**
MemoryMgr.h

**Prototype**
MemHandle MemHandleNew (UInt32 size)

**Parameters**
- `size` The desired size of the chunk.

**Result**
Returns a handle to the new chunk, or 0 if unsuccessful.

**Comments**
Use this call to allocate dynamic memory. Before you can write data to the memory chunk that `MemHandleNew` allocates, you must call `MemHandleLock` to lock the chunk and get a pointer to it.

**See Also**
MemPtrFree, MemPtrNew, MemHandleFree, MemHandleLock

MemHandleResize

**Purpose**
Resize a chunk.

**Declared In**
MemoryMgr.h

**Prototype**
Err MemHandleResize (MemHandle h, UInt32 newSize)

**Parameters**
- `h` Chunk handle.
- `newSize` The new desired size.

**Result**
0 No error.
memErrInvalidParam Invalid parameter passed.
memErrNotEnoughSpace Not enough free space in heap to grow chunk.
memErrChunkLocked
  Can’t grow chunk because it’s locked.

Comments  Call this routine to resize a chunk. This routine is always successful when shrinking the size of a chunk, even if the chunk is locked. When growing a chunk, it first attempts to grab free space immediately following the chunk so that the chunk does not have to move. If the chunk has to move to another free area of the heap to grow, it must be movable and have a lock count of 0.

On devices running version 2.0 or earlier of Palm OS, the MemHandleResize function tries to resize the chunk only within the same heap, whereas DmResizeRecord will look for space in other data heaps if it can’t find enough space in the original heap.

See Also  MemHandleNew, MemHandleSize

**MemHandleSetOwner**

Purpose  Set the owner ID of a chunk.

Declared In  MemoryMgr.h

Prototype  Err MemHandleSetOwner (MemHandle h, Uint16 owner)

Parameters  
- `h`  Chunk handle.
- `owner`  New owner ID of the chunk. Specify 0 to set the owner to the operating system.

Result  Returns 0 if no error, or memErrInvalidParam if an error occurs.

Comments  When you allocate a parameter block to pass to SysUIAppSwitch or SysAppLaunch, you must call MemPtrSetOwner to grant ownership of the parameter block chunk to the OS (your application is originally set as the owner). If the parameter block structure references any chunks by handle, you also must call MemHandleSetOwner to grant ownership of those blocks to the OS. If you don’t change the ownership of these chunks, they will get...
freed before the application you’re launching has a chance to use them.

**MemHandleSize**

**Purpose**  Return the requested size of a chunk.

**Declared In**  MemoryMgr.h

**Prototype**  UInt32 MemHandleSize (MemHandle h)

**Parameters**  

- **h**  Chunk handle.

**Result**  Returns the requested size of the chunk.

**Comments**  Call this routine to get the size originally requested for a chunk.

**See Also**  MemHandleResize

**MemHandleToLocalID**

**Purpose**  Convert a handle into a local chunk ID which is card relative.

**Declared In**  MemoryMgr.h

**Prototype**  LocalID MemHandleToLocalID (MemHandle h)

**Parameters**  

- **h**  Chunk handle.

**Result**  Returns local ID, or NULL (0) if unsuccessful.

**Comments**  Call this routine to convert a chunk handle to a local ID.

**See Also**  MemLocalIDToLocalIDToGlobal, MemLocalIDToLockedPtr
**MemHandleUnlock**

**Purpose**
Unlock a chunk given a chunk handle.

**Declared In**
MemoryMgr.h

**Prototype**
Err MemHandleUnlock (MemHandle h)

**Parameters**
- **h**
The chunk handle.

**Result**
- **0**
  No error.
- **memErrInvalidParam**
  Invalid parameter passed.

**Comments**
Call this routine to decrement the lock count for a chunk.
MemHandleLock and MemHandleUnlock should be used in pairs.

**See Also**
MemHandleLock

---

**MemHeapCheck**

**Purpose**
Check validity of a given heap.

**Declared In**
MemoryMgr.h

**Prototype**
Err MemHeapCheck (UInt16 heapID)

**Parameters**
- **heapID**
  ID of heap to check.

**Result**
Returns 0 if no error.

**See Also**
MemDebugMode, MemSetDebugMode
MemHeapCompact

Purpose: Compact a heap.

Declared In: MemoryMgr.h

Prototype: Err MemHeapCompact (UInt16 heapID)

Parameters:
- heapID: ID of the heap to compact.

Result: Always returns 0.

Comments: Most applications never need to call this function explicitly. The system software calls this function at various times; for example, during memory allocation (if sufficient free space is not available) and during system reboot.

Call this routine to compact a heap and merge all free space. This routine attempts to move all movable chunks to the start of the heap and merge all free space in the center of the heap.

MemHeapDynamic

Purpose: Return true if the given heap is a dynamic heap.

Declared In: MemoryMgr.h

Prototype: Boolean MemHeapDynamic (UInt16 heapID)

Parameters:
- heapID: ID of the heap to be tested.

Result: Returns true if dynamic, false if not.

Comments: Dynamic heaps are used for volatile storage, application stacks, globals, and dynamically allocated memory.
NOTE: In Palm OS 3.5, the dynamic heap is sized based on the amount of memory available, and is generally larger than before.

See Also MemNumHeaps, MemHeapID

MemHeapFlags

Purpose Return the heap flags for a heap.

Declared In MemoryMgr.h

Prototype UInt16 MemHeapFlags (UInt16 heapID)

Parameters -> heapID ID of heap.

Result Returns the heap flags.

Comments Call this routine to retrieve the heap flags for a heap. The flags can be examined to determine if the heap is ROM based or not. ROM-based heaps have the memHeapFlagReadOnly bit set (the memHeapFlagReadOnly mask has a value of 0x0001).

See Also MemNumHeaps, MemHeapID

MemHeapFreeBytes

Purpose Return the total number of free bytes in a heap and the size of the largest free chunk in the heap.

Declared In MemoryMgr.h

Prototype Err MemHeapFreeBytes (UInt16 heapID, UInt32* freeP, UInt32* maxP)

Parameters -> heapID ID of heap.
Memory Manager
Memory Manager Functions

<- freeP Pointer to a variable of type UInt32 for free bytes.
<- maxP Pointer to a variable of type UInt32 for max free chunk size. Do not pass NULL for this argument.

Result Always returns 0.

Comments This routine doesn’t compact the heap but may be used to determine in advance whether an allocation request will succeed. Before allocating memory, call this function and test the value returned in maxP to determine whether enough free space to fulfill your allocation request exists. If not, you may make more space available by calling the MemHeapCompact function. Note that both MemPtrNew and MemHandleNew automatically compact the heap if necessary.

See Also MemHeapSize, MemHeapID, MemHeapCompact

MemHeapID

Purpose Return the heap ID for a heap, given its index and the card number.

Declared In MemoryMgr.h

Prototype UInt16 MemHeapID (UInt16 cardNo, UInt16 heapIndex)

Parameters -> cardNo The card number, either 0 or 1.
-> heapIndex The heap index, anywhere from 0 to MemNumHeaps - 1.

Result Returns the heap ID.

Comments Call this routine to retrieve the heap ID of a heap, given the heap index and the card number. A heap ID must be used to obtain
information on a heap such as its size, free bytes, etc., and is also passed to any routines which manipulate heaps.

See Also MemNumHeaps

MemHeapScramble

Purpose Scramble the specified heap.

Declared In MemoryMgr.h

Prototype Err MemHeapScramble (UInt16 heapID)

Parameters heapID ID of heap to scramble.

Comments The system attempts to move each movable chunk. Useful for debugging.

Result Always returns 0.

See Also MemDebugMode, MemSetDebugMode

MemHeapSize

Purpose Return the total size of a heap including the heap header.

Declared In MemoryMgr.h

Prototype UInt32 MemHeapSize (UInt16 heapID)

Parameters -> heapID ID of heap.

Result Returns the total size of the heap.

See Also MemHeapFreeBytes, MemHeapID
MemLocalIDKind

**Purpose**
Return whether or not a local ID references a handle or a pointer.

**Declared In**
MemoryMgr.h

**Prototype**
LocalIDKind MemLocalIDKind (LocalID local)

**Parameters**
- `local` Local ID to query

**Result**
Returns LocalIDKind, or a `memIDHandle` or `memIDPtr` (see MemoryMgr.h).

**Comments**
This routine determines if the given local ID is to a nonmovable (`memIDPtr`) or movable (`memIDHandle`) chunk.

MemLocalIDToGlobal

**Purpose**
Convert a local ID, which is card relative, into a global pointer in the designated card.

**Declared In**
MemoryMgr.h

**Prototype**
MemPtr MemLocalIDToGlobal (LocalID local, UInt16 cardNo)

**Parameters**
- `local` The local ID to convert.
- `cardNo` Memory card the chunk resides in.

**Result**
Returns pointer or handle to chunk.

**See Also**
MemLocalIDKind, MemLocalIDToLockedPtr
**MemLocalIDToLockedPtr**

**Purpose**
Return a pointer to a chunk given its local ID and card number.
If the local ID references a movable chunk handle, this routine automatically locks the chunk before returning.

**Declared In**
MemoryMgr.h

**Prototype**
MemPtr MemLocalIDToLockedPtr (LocalID local, UInt16 cardNo)

**Parameters**
- `local` Local chunk ID.
- `cardNo` Card number.

**Result**
Returns pointer to chunk, or 0 if an error occurs.

**See Also**
MemLocalIDToGlobal, MemLocalIDToPtr, MemLocalIDKind, MemPtrToLocalID, MemHandleToLocalID

**MemLocalIDToPtr**

**Purpose**
Return pointer to chunk, given the local ID and card number.

**Declared In**
MemoryMgr.h

**Prototype**
MemPtr MemLocalIDToPtr (LocalID local, UInt16 cardNo)

**Parameters**
- `local` Local ID to query.
- `cardNo` Card number the chunk resides in.

**Result**
Returns a pointer to the chunk, or 0 if error.

**Comments**
If the local ID references a movable chunk and that chunk is not locked, this function returns 0 to indicate an error.

**See Also**
MemLocalIDToGlobal, MemLocalIDToLockedPtr
**Memory Manager**

**Memory Manager Functions**

---

**MemMove**

**Purpose**
Move a range of memory to another range in the dynamic heap.

**Declared In**
MemoryMgr.h

**Prototype**
Err MemMove (void* dstP, const void* sP, Int32 numBytes)

**Parameters**
- dstP Pointer to destination.
- sP Pointer to source.
- numBytes Number of bytes to move.

**Result**
Always returns 0.

**Comments**
Handles overlapping ranges.
For operations where the destination is in a data heap, see [DmSet](#), [DmWrite](#), and related functions.

---

**MemNumCards**

**Purpose**
Return the number of memory card slots in the system. Not all slots need to be populated.

**Declared In**
MemoryMgr.h

**Prototype**
UInt16 MemNumCards (void)

**Parameters**
None.

**Result**
Returns number of slots in the system.
MemNumHeaps

Purpose  Return the number of heaps available on a particular card.

Declared In  MemoryMgr.h

Prototype  UInt16 MemNumHeaps (UInt16 cardNo)

Parameters  -> cardNo  The card number; either 0 or 1.

Result  Number of heaps available, including ROM- and RAM-based heaps.

Comments  Call this routine to retrieve the total number of heaps on a memory card. The information can be obtained by calling MemHeapSize, MemHeapFreeBytes, and MemHeapFlags on each heap using its heap ID. The heap ID is obtained by calling MemHeapID with the card number and the heap index, which can be any value from 0 to MemNumHeaps.

MemNumRAMHeaps

Purpose  Return the number of RAM heaps in the given card.

Declared In  MemoryMgr.h

Prototype  UInt16 MemNumRAMHeaps (UInt16 cardNo)

Parameters  cardNo  The card number.

Result  Returns the number of RAM heaps.

See Also  MemNumCards
Memory Manager
Memory Manager Functions

MemPtrCardNo

Purpose
Return the card number (0 or 1) a nonmovable chunk resides on.

Declared In
MemoryMgr.h

Prototype
UInt16 MemPtrCardNo (MemPtr p)

Parameters
-> p Pointer to the chunk.

Result
Returns the card number.

See Also
MemHandleCardNo

MemPtrDataStorage

Purpose
Return true if the given pointer is part of a data storage heap; if not, it is a pointer in the dynamic heap.

Declared In
MemoryMgr.h

Prototype
Boolean MemPtrDataStorage (MemPtr p)

Parameters
p Pointer to a chunk.

Result
Returns true if the chunk is part of a data storage heap.

Comments
Called by Fields package to determine if it needs to worry about data storage write-protection when editing a text field.

See Also
MemHeapDynamic
MemPtrFree

Purpose  Macro to dispose of a chunk.

Declared In  MemoryMgr.h

Prototype  Err MemPtrFree (MemPtr p)

Parameters  - > p  Pointer to a chunk.

Result  0  If no error or memErrInvalidParam (invalid parameter).

Comments  Call this routine to dispose of a nonmovable chunk.

MemPtrHeapID

Purpose  Return the heap ID of a chunk.

Declared In  MemoryMgr.h

Prototype  UInt16 MemPtrHeapID (MemPtr p)

Parameters  - > p  Pointer to the chunk.

Result  Returns the heap ID of a chunk.

Comments  Call this routine to get the heap ID of the heap a chunk resides in.
**MemPtrNew**

**Purpose**
Allocate a new nonmovable chunk in the dynamic heap.

**Declared In**
MemoryMgr.h

**Prototype**
MemPtr MemPtrNew (UInt32 size)

**Parameters**
- *size*  
The desired size of the chunk.

**Result**
Returns pointer to the new chunk, or 0 if unsuccessful.

**Comments**
This routine allocates a nonmovable chunk in the dynamic heap and returns a pointer to the chunk. Applications can use it when allocating dynamic memory. Note that chunks range in size from 1 byte to slightly less than 64KB; you cannot allocate a single chunk larger than this.

In Palm OS 3.5, the dynamic heap is sized based on the amount of memory available, and is generally larger than before.

**NOTE:** You cannot allocate a zero-size reference block.

**MemPtrRecoverHandle**

**Purpose**
Recover the handle of a movable chunk, given a pointer to its data.

**Declared In**
MemoryMgr.h

**Prototype**
MemHandle MemPtrRecoverHandle (MemPtr p)

**Parameters**
- *p*  
Pointer to the chunk.

**Result**
Returns the handle of the chunk, or 0 if unsuccessful.

**Comments**
Don’t call this function for pointers in ROM or nonmovable data chunks.
MemPtrResize

**Purpose**
Resize a chunk.

**Declared In**
MemoryMgr.h

**Prototype**
```
Err MemPtrResize (MemPtr p, UInt32 newSize)
```

**Parameters**
- `p` Pointer to the chunk.
- `newSize` The new desired size.

**Result**
Returns 0 if no error, or `memErrNotEnoughSpace` `memErrInvalidParam`, or `memErrChunkLocked` if an error occurs.

**Comments**
Call this routine to resize a locked chunk. This routine is always successful when shrinking the size of a chunk. When growing a chunk, it attempts to use free space immediately following the chunk.

**See Also**
MemPtrSize, MemHandleResize

MemPtrSetOwner

**Purpose**
Set the owner ID of a chunk.

**Declared In**
MemoryMgr.h

**Prototype**
```
Err MemPtrSetOwner (MemPtr p, UInt16 owner)
```

**Parameters**
- `p` Pointer to the chunk.
- `owner` New owner ID of the chunk. Specify 0 to set the owner to the operating system.

**Result**
Returns 0 if no error, or `memErrInvalidParam` if an error occurs.
Comments

When you allocate a parameter block to pass to SysUIAppSwitch or SysAppLaunch, you must call MemPtrSetOwner or MemHandleSetOwner to grant ownership of the parameter block chunk, and any other chunks referenced in it, to the OS (your application is originally set as the owner). If you don’t change the ownership of the parameter block, it will get freed before the application you’re launching has a chance to use it.

MemPtrSize

Purpose

Return the size of a chunk.

Declared In

MemoryMgr.h

Prototype

UInt32 MemPtrSize (MemPtr p)

Parameters

- -> p Pointer to the chunk.

Result

The requested size of the chunk.

Comments

Call this routine to get the original requested size of a chunk.

MemPtrToLocalID

Purpose

Convert a pointer into a card-relative local chunk ID.

Declared In

MemoryMgr.h

Prototype

LocalID MemPtrToLocalID (MemPtr p)

Parameters

- -> p Pointer to a chunk.

Result

Returns the local ID of the chunk.

Comments

Call this routine to convert a chunk pointer to a local ID.

See Also

MemLocalIDToPtr
MemPtrUnlock

**Purpose**  Unlock a chunk, given a pointer to the chunk.

**Declared In**  MemoryMgr.h

**Prototype**  Err MemPtrUnlock (MemPtr p)

**Parameters**

- **p**  Pointer to a chunk.

**Result**  0 if no error, or memErrInvalidParam if an error occurs.

**Comments**  A chunk must **not** be unlocked more times than it was locked.

**See Also**  MemHandleLock

MemSet

**Purpose**  Set a memory range in a dynamic heap to a specific value.

**Declared In**  MemoryMgr.h

**Prototype**  Err MemSet (void* dstP, Int32 numBytes, UInt8 value)
Memory Manager
Memory Manager Functions

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstP</td>
<td>Pointer to the destination.</td>
</tr>
<tr>
<td>numBytes</td>
<td>Number of bytes to set.</td>
</tr>
<tr>
<td>value</td>
<td>Value to set.</td>
</tr>
</tbody>
</table>

Result
Always returns 0.

Comments
For operations where the destination is in a data heap, see DmSet, DmWrite, and related functions.

MemSetDebugMode

Purpose
Set the debugging mode of the memory manager.

Declared In
MemoryMgr.h

Prototype
Err MemSetDebugMode (UInt16 flags)

Parameters
flags Debug flags.

Comments
Use the logical OR operator (|) to provide any combination of one, more, or none of the following flags:

- memDebugModeCheckOnChange
- memDebugModeCheckOnAll
- memDebugModeScrambleOnChange
- memDebugModeScrambleOnAll
- memDebugModeFillFree
- memDebugModeAllHeaps
- memDebugModeRecordMinDynHeapFree

Result
Returns 0 if no error, or -1 if an error occurs.
MemStoreInfo

Purpose
Return information on either the RAM store or the ROM store for a memory card.

Declared In
MemoryMgr.h

Prototype
Err MemStoreInfo (UInt16 cardNo,
UInt16 storeNumber, UInt16* versionP,
UInt16* flagsP, Char* nameP, UInt32* crDateP,
UInt32* bckUpDateP, UInt32* heapListOffsetP,
UInt32* initCodeOffset1P,
UInt32* initCodeOffset2P, LocalID* databaseDirIDP)

Parameters
-> cardNo Card number, either 0 or 1.
-> storeNumber Store number; 0 for ROM, 1 for RAM.
<< versionP Pointer to version variable, or 0.
<< flagsP Pointer to flags variable, or 0.
<< nameP Pointer to character array (32 bytes), or 0.
<< crDateP Pointer to creation date variable, or 0.
<< bckUpDateP Pointer to backup date variable, or 0.
<< heapListOffsetP Pointer to heapListOffset variable, or 0.
<< initCodeOffset1P Pointer to initCodeOffset1 variable, or 0.
<< initCodeOffset2P Pointer to initCodeOffset2 variable, or 0.
<< databaseDirIDP Pointer to database directory chunk ID variable, or 0.

Result
Returns 0 if no error, or memErrCardNotPresent,
memErrRAMOnlyCard, or memErrInvalidStoreHeader if an error occurs.
Comments  Call this routine to retrieve any or all information on either the RAM store or the ROM store for a card. Pass 0 for variables that you don’t wish returned.
This chapter describes the Notification Manager API as declared in the header file NotifyMgr.h. It discusses the following topics:

- Notification Constants
- Notification Functions
- Application-Defined Functions

The chapter “Notifications” in this book lists the possible notifications and describes the data sent with each. Also see the section “Notifications” on page 30 in the “Application Startup and Stop” chapter of the Palm OS Programmer’s Companion, vol. I for a description of how to use notifications.

### Notification Constants

#### Miscellaneous Constants

This following miscellaneous constants are used in the Notification Manager. For other notification constants, see the “Notifications” chapter in this book.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysNotifyBroadcasterCode</td>
<td>'psys'</td>
<td>The value passed as the creator ID of the broadcaster for notifications broadcast by the system.</td>
</tr>
<tr>
<td>sysNotifyDefaultQueueSize</td>
<td>30</td>
<td>The maximum number of nested broadcasts allowed.</td>
</tr>
<tr>
<td>sysNotifyNoDatabaseID</td>
<td>0xFFFFFFFF</td>
<td>The database local ID used by the system when it registers for notifications.</td>
</tr>
</tbody>
</table>
**Notification Manager**

**Notification Functions**

---

### SysNotifyBroadcast

**Purpose**
Synchronously send a notification to all applications registered for it.

**Declared In**
NotifyMgr.h

**Prototype**
```
Err SysNotifyBroadcast
    (SysNotifyParamType *notify)
```

**Parameters**
- `notify`: Identifies the notification to be broadcast. See `SysNotifyParamType`.

**Result**
Returns one of the following error codes:
- `errNone`: No error.
- `sysNotifyErrBroadcastBusy`: The broadcast stack limit has already been reached.
- `sysErrParamErr`: The background thread is broadcasting the notification and the `notify` parameter is NULL.
- `sysNotifyErrNoStackSpace`: There is not enough space on the stack for the notification.

---

### Constant Value Description

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysNotifyNormalPriority</td>
<td>0</td>
<td>Typical priority value used when registering for notifications.</td>
</tr>
<tr>
<td>sysNotifyVersionNum</td>
<td>1</td>
<td>Current Notification Manager version. This number is stored in the system feature <code>sysFtrNumNotifyMgrVersion</code>.</td>
</tr>
</tbody>
</table>

---

802  *Palm OS Programmer’s API Reference*
Comments

When you call this function, the notification you specify is broadcast to all applications, shared libraries, and other code resources that have registered to receive that notification. The broadcast is performed synchronously, meaning that the system broadcasts the notification immediately and waits for each notification client to perform its notification handler and return before the SysNotifyBroadcast call returns. This notification occurs in priority order.

The system allows nested notifications; that is, the recipient of a notification might broadcast a new notification, whose recipient might broadcast another new notification and so on. The constant sysNotifyDefaultQueueSize specifies how many levels of nested notification are allowed. If you reach this limit, the error sysNotifyErrBroadcastBusy is returned and your notification is not broadcast. To avoid reaching the limit, use SysNotifyBroadcastDeferred instead of SysNotifyBroadcast in your notification handlers. This ensures that the notification will not be broadcast until the top of the event loop.

**WARNING!** Do not call SysNotifyBroadcast from code that might be called from a background task (such as a trap patch) with the memory semaphore reserved. Use SysNotifyBroadcastDeferred instead.

Compatibility

Implemented only if Notification Feature Set is present.
SysNotifyBroadcastDeferred

Purpose
Enqueue a notification for later broadcast.

Declared In
NotifyMgr.h

Prototype
Err SysNotifyBroadcastDeferred
(SysNotifyParamType *notify, Int16 paramSize)

Parameters
<- notify The notification to enqueue. See SysNotifyParamType.
-> paramSize Size of the data pointed to by the field notify->notifyDetailsP.

Result
Returns one of the following error codes:
errNone No error.
memErrNotEnoughSpace There is not enough memory to allocate a new notification entry in the queue.
sysErrParamErr paramSize is a negative number.
sysNotifyErrQueueFull The queue has reached its maximum number of entries.

Comments
This function is similar to SysNotifyBroadcast except that the broadcast does not take place until the top of the event loop (specifically, the next time EvtGetEvent is called). The system copies the notify structure to a new memory chunk, which is disposed of upon completion of the broadcast. (The paramSize value is used when copying the notifyDetailsP portion of the notify structure.)

Compatibility
Implemented only if Notification Feature Set is present.
SysNotifyBroadcastFromInterrupt

**Purpose**  Allows interrupt handlers to enqueue a notification for later broadcast.

**Declared In**  NotifyMgr.h

**Prototype**  
```c
Err SysNotifyBroadcastFromInterrupt
(UInt32 notifyType, UInt32 broadcaster,
void *notifyDetailsP)
```

**Parameters**
- `-> notifyType`  The type of event that occurred. See the chapter Notifications for a complete list of the notifications that Palm OS® broadcasts.
- `-> broadcaster`  The creator ID of the device or application that broadcast the notification.
- `-> notifyDetailsP`  Pointer to data specific to this notification. See the Notifications chapter for the specific instances where this parameter is used.

**Result**  Returns one of the following error codes:
- `errNone`  No error.
- `sysNotifyErrQueueFull`  The queue has reached its maximum number of entries.

**Comments**  Like SysNotifyBroadcastDeferred, this function enqueues a notification to be broadcast at the top of the event loop (specifically, the next time EvtGetEvent is called). It differs from SysNotifyBroadcastDeferred in that it is interrupt-safe and intended to be called from interrupt handlers.

This function is intended to be used by device drivers and other low-level software to generate a notification about a hardware change. For example, the Expansion Manager uses SysNotifyBroadcastFromInterrupt to broadcast sysNotifyCardInsertedEvent and
sysNotifyCardRemovedEvent when a card is inserted into or removed from the expansion slot.
SysNotifyBroadcastFromInterrupt is not intended to be used by general third party applications. Patching SysNotifyBroadcastFromInterrupt will cause the system to hang.

Compatibility Implemented only if 4.0 New Feature Set is present.

SysNotifyRegister

Purpose Register to receive a notification.

Declared In NotifyMgr.h

Prototype Err SysNotifyRegister (UInt16 cardNo, LocalID dbID, UInt32 notifyType, SysNotifyProcPtr callbackP, Int8 priority, void *userDataP)

Parameters

-> cardNo Number of the storage card on which the application or code resource resides.
-> dbID Local ID of the application or code resource.
-> notifyType The notification that the application wants to receive. See the chapter Notifications.
-> callbackP Set to NULL to receive the notification as an application launch code. If your code does not have a PilotMain function (for example, if it is a shared library), pass a pointer to a function that should be called when the notification is broadcast. See SysNotifyProcPtr.
The priority with which the application should receive the event. Most applications and other code resources should always use \texttt{sysNotifyNormalPriority}. In rare circumstances, you may need to ensure that your code is notified toward the beginning or toward the end of the notification sequence. If so, be sure to leave some space so that your code won’t collide with the system’s handling of notifications or with another application’s handling of notifications. In general, \texttt{Palm}™ recommends using a value whose least significant bits are 0 (such as 32, 64, 96, and so on). The smaller the priority, the earlier your code is notified.

\texttt{userDataP} Caller-defined data to pass to the notification handler.

\textbf{Result} Returns one of the following error codes:

- \texttt{errNone} No error.
- \texttt{sysErrParamErr} The database ID is \texttt{NULL}.
- \texttt{sysNotifyErrDuplicateEntry} This application is already registered to receive this notification.

\textbf{Comments} Call this function when your code should receive a notification that a specific event has occurred or is about to occur. See the \texttt{Notifications} chapter for a list of the possible notifications. Once you register for a notification, you remain registered to receive it until a system reset occurs or until you explicitly unregister using \texttt{SysNotifyUnregister}.

If you’re writing an application, you should pass \texttt{NULL} as the \texttt{callbackP} parameter. In this case, the system notifies your application by sending it the \texttt{sysAppLaunchCmdNotify} launch code. This launch code’s parameter block points to a \texttt{SysNotifyParamType} structure containing details about the notification.
If your code is not in an application, for example, it is a shared library or a separately loaded code resource, then receiving a launch code is not possible. In this case, pass a pointer to a callback function in `callbackP`. This callback should follow the prototype shown in `SysNotifyProcPtr`. Note that you should always supply a card number and database ID to `SysNotifyRegister`, even if you specify a callback function.

**IMPORTANT:** Because the `callbackP` pointer is used to directly call the function, the pointer must remain valid from the time `SysNotifyRegister` is called to the time the notification is broadcast. If the function is in a shared library, you must keep the library open. If the function is in a separately loaded code resource, the resource must remain locked while registered for the notification. When you close a library or unlock a resource, you must first unregister for any notifications. If you don’t, the system will crash when the notification is broadcast.

Whether the notification handler is responding to `sysAppLaunchCmdNotify` or uses the callback function, the notification handler may perform any processing necessary. As with most launch codes, it’s not possible to access global variables. If the handler needs access to any particular value to respond to the notification, pass a pointer to that value in the `userDataP` parameter. The system passes this pointer back to your application or callback function in the launch code’s parameter block.

The notification handler may unregister for this notification or register for other notifications. It may also broadcast another notifications; however, it’s recommended that you use `SysNotifyBroadcastDeferred` to do this so as not to overflow the broadcast stack.

You may display a user interface in your notification handler; however, you should be careful when you do so. Many of the notifications are broadcast during `SysHandleEvent`, which means your application event loop might not have progressed to the point where it is possible for you to display a user interface, or you might overflow the stack by displaying a user interface at this stage. See
the “Notifications” chapter to learn which notifications are broadcast during SysHandleEvent.

**Compatibility**

Implemented only if Notification Feature Set is present.

## SysNotifyUnregister

**Purpose**

Cancel notification of the given event.

**Declared In**

NotifyMgr.h

**Prototype**

```
Err SysNotifyUnregister(UInt16 cardNo, LocalID dbID, UInt32 notifyType, Int8 priority)
```

**Parameters**

- **-> cardNo**  Number of the storage card on which the application or code resource resides.
- **-> dbID**  Local ID of the application or code resource.
- **-> notifyType**  The notification for which to unregister. See Notifications.
- **-> priority**  The priority value you passed as part of SysNotifyRegister.

**Result**

Returns one of the following error codes:

- **errNone**  No error.
- **sysNotifyErrEntryNotFound**  The application wasn’t registered to receive this notification.

**Comments**

Use this function to remove your code from the list of those that receive notifications about a particular event. This function is particularly necessary if you are writing a shared library or a separately loaded code resource that receives notifications. When the resource is unloaded, it must unregister for all of its notifications, or the system will crash when the notification is broadcast.
Compatibility
 Implemented only if Notification Feature Set is present.

Application-Defined Functions

SysNotifyProcPtr

Purpose
 Handle a notification.

Declared In
 NotifyMgr.h

Prototype
 Err (*SysNotifyProcPtr) (SysNotifyParamType *notifyParamsP)

Parameters
 -> notifyParamsP
 Pointer to a structure that contains the notification event that occurred and any other information about it. See SysNotifyParamType.

Result
 Always return 0.

Comments
 You pass this function as a parameter to SysNotifyRegister when registering code that does not have a PilotMain for a notification. See the description of SysNotifyRegister for advice on writing a notification handler.

IMPORTANT:
 Because the callbackP pointer is used to directly call the function, the pointer must remain valid from the time SysNotifyRegister is called to the time the notification is broadcast. If the function is in a shared library, you must keep the library open. If the function is in a separately loaded code resource, the resource must remain locked while registered for the notification. When you close a library or unlock a resource, you must first unregister for any notifications. If you don’t, the system will crash when the notification is broadcast.
Overlay Manager

This chapter describes the overlay manager API as declared in the header file OverlayMgr.h. It discusses the following topics:

- Overlay Manager Data Structures
- Overlay Manager Constants
- Overlay Manager Functions

For more information on the overlay manager, see “Using Overlays to Localize Resources” on page 365 of the Palm OS Programmer’s Companion, vol. I.

Overlay Manager Data Structures

OmLocaleType

The OmLocaleType struct specifies a locale.

```c
typedef struct {
    UInt16   language;
    UInt16   country;
} OmLocaleType;
```

Field Descriptions

- **language** The language spoken in the locale. This value is one of the LanguageType constants.
- **country** The country or region where the language is spoken. This value is one of the CountryType constants.

Compatibility

If Palm OS® 4.0 New Feature Set is present, the LmLocaleType replaces OmLocaleType. For backward compatibility, OmLocaleType is mapped to LmLocaleType.
# Overlay Manager

## Overlay Manager Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omOverlayRscType</td>
<td>'ovly'</td>
<td>Symbolic name of an overlay resource that is contained in both the base database and the overlay database.</td>
</tr>
<tr>
<td>omOverlayRscID</td>
<td>1000</td>
<td>Resource ID of the overlay resource in both the base database and the overlay database.</td>
</tr>
<tr>
<td>omFtrCreator</td>
<td>'ovly'</td>
<td>Creator value used for the omFtrShowErrorsFlag feature.</td>
</tr>
<tr>
<td>omFtrDefaultLocale</td>
<td>1</td>
<td>Feature that specifies the default locale stored in the ROM. The default locale is used in cases where the system is attempting to open a “stripped” database (one that requires an overlay) and an overlay matching the current locale cannot be found. In this case, the system then looks for an overlay matching the default locale. Use FtrGet and FtrSet to retrieve and set this value.</td>
</tr>
<tr>
<td>omFtrShowErrorsFlag</td>
<td>0</td>
<td>Feature that controls the number of error messages displayed by the overlay manager. If this feature is set to true, the overlay manager may display several more error messages when validating an overlay against its base database. This feature only takes effect when the error checking level is set to full (common on debug ROMs, not on release ROMs). Use FtrGet and FtrSet to retrieve and set this value.</td>
</tr>
</tbody>
</table>
Overlay Manager Functions

OmGetCurrentLocale

Purpose
Return the current locale.

Declared In
OverlayMgr.h

Prototype
void OmGetCurrentLocale
(LmLocaleType *currentLocale)

Parameters
<- currentLocale
  Points to an LmLocaleType struct that identifies the current locale.

Result
Returns nothing.

Comments
This function returns the current locale. The current locale controls which overlays are used for resource databases. For example, suppose you have one application and two associated overlays installed, one for US English and one for British English. In this case, if the country specified in the locale returned by this function is cUnitedKingdom, the British English overlay is used for the application. If the country returned is cUnitedStates, the US English overlay is used.

Compatibility
Implemented only if 3.5 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call OmGlueGetCurrentLocale. For more information, see Chapter 75, “PalmOSGlue Library.”

See Also
OmGetSystemLocale
OmGetIndexedLocale

Purpose

Return a system locale by index.

Declared In

OverlayMgr.h

Prototype

Err OmGetIndexedLocale (UInt16 localeIndex, LmLocaleType *theLocale)

Parameters

--> localeIndex Zero-based index of the locale to return.
<- theLocale Points to an LmLocaleType struct that identifies the locale at that index.

Result

Returns errNone upon success, or omErrInvalidLocaleIndex if the index is out of bounds.

Comments

OmGetIndexedLocale is used in a loop to discover how many system overlays are installed for system resources.

If the 4.0 New Feature Set is present, use OmGetNextSystemLocale instead of this function.

OmGetIndexedLocale can be slow on ROMs that contain many valid system locales.

Compatibility

Implemented only if 3.5 New Feature Set is present.

In Palm OS 3.5, this function would not return a system overlay that was located in RAM. The Palm OS 4.0 version of this function does return system overlays located in RAM.

See Also

OmGetSystemLocale, OmGetNextSystemLocale
OmGetNextSystemLocale

**Purpose**  
Return a system locale.

**Declared In**  
OverlayMgr.h

**Prototype**  
Err OmGetNextSystemLocale (Boolean iNewSearch,  
OmSearchStateType *ioStateInfoP,  
LmLocaleType *oLocaleP)

**Parameters**  
- `<-> iNewSearch`  
  true if this function call is starting a new search, or false if this function call is a continuation of a search.

- `<-> ioStateInfoP`  
  If `iNewSearch` is false, this must point to the same data used for the previous invocation.

- `<- oLocaleP`  
  The found locale.

**Result**  
Returns errNone if no error or omErrNoNextSystemLocale if no matches were found.

**Comments**  
You can call this function successively to discover how many system overlays are installed for system resources. Each system overlay found determines a separate valid system locale. Any locale returned by this function can be sent to OmSetSystemLocale to change the system locale.

To start the search, pass true for `iNewSearch`. Allocate an OmSearchStateType structure and pass its address as `ioStateInfoP`. OmGetNextSystemLocale stores private information in `ioStateInfoP` and uses it if the search is continued.

To continue a search where the previous one left off, pass false for `iNewSearch` and pass the same `ioStateInfoP` that you used during the previous call to this function.

When called successively, this function eventually returns all system overlays that are in ROM or RAM.
Compatibility Implemented only if 4.0 New Feature Set is present.

OmGetRoutineAddress

Purpose Return the address of an overlay manager function.

Declared In OverlayMgr.h

Prototype void *OmGetRoutineAddress (OmSelector inSelector)

Parameters -> inSelector One of the routine selectors defined in OverlayMgr.h.

Result Returns the address of the corresponding function. Returns NULL if an invalid routine selector is passed.

Comments You typically use this function to determine whether an overlay manager function exists on the device. As future releases of Palm OS add new functions, older devices with earlier versions of the overlay manager will not implement these newer functions. If OmGetRoutineAddress returns NULL, the function is unavailable.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also IntlGetRoutineAddress, SysGetTrapAddress
OmGetSystemLocale

**Purpose**
Return the system locale.

**Declared In**
OverlayMgr.h

**Prototype**
```c
void OmGetSystemLocale
(LmLocaleType *systemLocale)
```

**Parameters**
`systemLocale` Points to an `LmLocaleType` struct that identifies the system locale.

**Result**
Returns nothing.

**Comments**
You typically don’t use this function. Instead, use `OmGetCurrentLocale`, which returns the locale that determines which overlays are used.

The system locale is saved in the storage heap header and persists across soft resets. When the device is reset, the system locale is used to set the current locale.

**Compatibility**
Implemented only if 3.5 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `OmGlueGetSystemLocale`. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**
`OmGetCurrentLocale`
OmLocaleToOverlayDBName

Purpose
Return the overlay database’s name given the base database name and the locale.

Declared In
OverlayMgr.h

Prototype
Err OmLocaleToOverlayDBName
(const Char *baseDBName,
const LmLocaleType *targetLocale,
Char *overlayDBName)

Parameters
- baseDBName
  The name of the base resource database associated with the overlay.
- targetLocale
  The locale to which this overlay applies. See LmLocaleType. Pass NULL to use the current locale.
- overlayDBName
  The overlay database name given the base database name and the target locale. This buffer must be at least dmDBNameLength bytes.

Result
Returns errNone upon success, or omErrUnknownLocale if the targetLocale parameter is invalid.

Comments
The appropriate overlay database name is currently:

baseDBName_1lCC

where:
baseDBName
  The name of the base database as you passed it in.
1l
  A two-character code identifying the language.
CC
  A two-character code identifying the country.

The base database name is truncated if necessary to allow for this suffix.
For example, the base database “MemoPad” might have an overlay for US English named “MemoPad_enUS”.

**Compatibility**  Implemented only if [3.5 New Feature Set](#) is present.

**See Also**  OmOverlayDBNameToLocale

---

### OmOverlayDBNameToLocale

**Purpose**  Return an overlay database’s locale given its name.

**Declared In**  OverlayMgr.h

**Prototype**

```c
Err OmOverlayDBNameToLocale
    (const Char *overlayDBName,
     LmLocaleType *overlayLocale)
```

**Parameters**

- `-> overlayDBName`  The name of the overlay database.
- `<- overlayLocale`  Points to an `LmLocaleType` structure identifying the overlay’s locale.

**Result**  Returns `errNone` upon success, `omErrBadOverlayDBName` if the string `overlayDBName` is not long enough to have a locale suffix, or `omErrUnknownLocale` if the locale cannot be determined.

**Compatibility**  Implemented only if [3.5 New Feature Set](#) is present.

**See Also**  OmLocaleToOverlayDBName
OmSetSystemLocale

**Purpose**
Set the system locale and reset the device.

**Declared In**
OverlayMgr.h

**Prototype**
Err OmSetSystemLocale
(const LmLocaleType *systemLocale)

**Parameters**
-> systemLocale  An LmLocaleType structure specifying the locale to switch the system to.

**Result**
Returns `errNone` upon success, or one of the following if an error occurs:
- `omErrUnknownLocale`
  There is no system overlay for `systemLocale`.
- `omErrInvalidLocale`
  The system overlay for `systemLocale` has been found but is invalid.
- `dmErrInvalidParam`
  An error occurred while opening the overlay.
- `dmErrMemError`
  A memory error occurred while opening the overlay.
- `dmErrDatabaseOpen`
  The system overlay was already open.

**Comments**
This function changes the system locale to the specified locale if it exists. It first determines that the system overlay exists for the requested locale and that it matches the base system database. If so, it updates the system locale information saved in the storage heap header and resets the device. After the device is reset, the current locale is set to the system locale.

A Palm Powered™ device has a default locale hard-coded into the ROM. This locale is used to set the system locale after a hard reset or any time that the storage heap header is invalid. The storage heap header is typically only invalid when the device is turned on for the first time.
Compatibility

Implemented only if 3.5 New Feature Set is present.

In Palm OS 3.5, this function would not switch to a system overlay that was located in the RAM. The Palm OS 4.0 version of this function does return system overlays located in the RAM.

See Also

OmGetSystemLocale
This chapter provides reference material for the password API. The password API is declared in the header file `Password.h`.

### Password Functions

**PwdExists**

**Purpose**
Return true if the system password is set.

**Declared In**
`Password.h`

**Prototype**
```c
Boolean PwdExists()
```

**Parameters**
None

**Result**
Returns true if the system password is set.

**PwdRemove**

**Purpose**
Remove the encrypted password string and recover data hidden in databases.

**Declared In**
`Password.h`

**Prototype**
```c
void PwdRemove (void)
```

**Parameters**
None

**Result**
Returns nothing.
Password
Password Functions

PwdSet

Purpose Use a passed string as the new password. The password is stored in an encrypted form.

Declared In Password.h

Prototype void PwdSet (Char* oldPassword, Char* newPassword)

Parameters oldPassword The old password must be successfully verified or the new password isn’t accepted
newPassword Char* to a string to use as the password. NULL means no password.

Result Returns nothing.

PwdVerify

Purpose Verify that the string passed matches the system password.

Declared In Password.h

Prototype Boolean PwdVerify (Char* string)

Parameters string String to compare to the system password. NULL means no current password.
The allocated length of string must be at least pwdLength characters long.

Result Returns true if the string matches the system password.
Pen Manager

This chapter provides reference material for the pen manager. The pen manager API is declared in the header file PenMgr.h.

For more information on the pen manager, see “The Pen Manager” on page 62 of the Palm OS Programmer’s Companion, vol. I.

Pen Manager Functions

PenCalibrate

Purpose
Set the calibration of the pen.

Declared In
PenMgr.h

Prototype
Err PenCalibrate (PointType* digTopLeftP, PointType* digBotRightP, PointType* scrTopLeftP, PointType* scrBotRightP)

Parameters
digTopLeftP Digitizer output from top-left coordinate.
digBotRightP Digitizer output from bottom-right coordinate.
scrTopLeftP Screen coordinate near top-left corner.
scrBotRightP Screen coordinate near bottom-right corner.

Result
Returns 0 if no error.

Comments
Called by Preferences application when calibrating pen.

See Also
PenResetCalibration
PenResetCalibration

**Purpose**  
Reset the calibration in preparation for calibrating the pen again.

**Declared In**  
PenMgr.h

**Prototype**  
Err PenResetCalibration (void)

**Parameters**  
None.

**Result**  
Always returns 0.

**Comments**  
Called by Preferences application before capturing points when calibrating the pen.

**See Also**  
PenCalibrate

---

**WARNING!**  
The digitizer is off after calling this routine and must be calibrated again!
Preferences

This chapter describes the preferences API as declared in the header file Preferences.h. It discusses the following topics:

- Preferences Data Types
- Preferences Constants
- Preferences Functions

For more information on preferences, see the section “Preferences” on page 320 of the *Palm OS Programmer’s Companion*, vol. I.

Preferences Data Types

**MeasurementSystemType**

The MeasurementSystemType enum specifies the preference for units of measure.

```c
typedef enum {
    unitsEnglish = 0,
    unitsMetric
} MeasurementSystemType;
```

**Value Descriptions**

- unitsEnglish: The English measurement system (feet, inches, and so on).
- unitsMetric: The Metric system (meters, centimeters, and so on).

**SecurityAutoLockType**

The SecurityAutoLockType enum specifies the values for the auto-locking preference. The auto-locking preference specifies when the device will shut down and lock itself.
typedef enum {
    never = 0,
    uponPowerOff,
    atPresetTime,
    afterPresetDelay
} SecurityAutoLockType;

Value Descriptions
never                  The auto-lock feature is disabled.
uponPowerOff          The device locks itself each time it is
                       powered off.
atPresetTime           The device locks itself at a certain time
                       every day.
afterPresetDelay      The device locks itself after a certain
                       amount of idle time.

SoundLevelTypeV20
The SoundLevelTypeV20 enum specifies whether certain sounds
are enabled or disabled. This enum is used as the values for several
of the system sound preferences.

typedef enum {
    s1On = 0,
    s1Off = 1
} SoundLevelTypeV20;

Value Descriptions
s1On                   Enabled
s1Off                  Disabled

Compatibility         This enumerated type is obsolete in Palm OS® versions 3.0 and
                       higher. The preferences that use this enum are replaced by
                       preferences that indicate sound volume as well as whether the
                       sound is on or off.
SystemPreferencesChoice

The SystemPreferencesChoice enum defines values that you can pass to PrefGetPreference and PrefSetPreference to retrieve or set a system preference value. SystemPreferencesChoice defines one constant for each field in the SystemPreferencesType structure, which should be considered a private structure.

Table 43.1 lists and describes the SystemPreferencesChoice constants. For each constant, it shows what type of data is returned by PrefGetPreference for that constant and which version of the system preferences structure contains this preference.

The system preferences structure keeps track of its own version information because it has been updated several times. Each Palm OS release that updates the system preferences structure increments the structure’s version number. Table 43.1 on page 830 specifies which version of the system preferences structure introduced that preference. For each preferences version, there is a constant representing that version. See the section “Preferences Constants” on page 839 for information on which preferences version corresponds with which Palm OS release. You should check the preference’s version number before attempting to obtain the value of any preference introduced after version 2. For example:

```c
LmLocaleType currentLocale;
if (PrefGetPreference(prefVersion) >= preferenceDataVer9) {
    currentLocale = (LmLocaleType) PrefGetPreference(prefLocale);
}
```

Most of the system preferences can be set in the Preferences and Security applications. Table 43.2 on page 838 specifies which system preference is set by each user interface field in these two applications.
### Table 43.1 SystemPreferencesChoice Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefVersion</td>
<td>UInt16</td>
<td>2</td>
<td>The preferences version number.</td>
</tr>
<tr>
<td>prefCountry</td>
<td>CountryType</td>
<td>2</td>
<td>The country for which the device was built.</td>
</tr>
<tr>
<td>prefDateFormat</td>
<td>DateFormatType</td>
<td>2</td>
<td>The short format used to display dates. For example: 95/12/31</td>
</tr>
<tr>
<td>prefLongDateFormat</td>
<td>DateFormatType</td>
<td>2</td>
<td>The long format used to display dates. For example: 31 Dec 1995</td>
</tr>
<tr>
<td>prefWeekStartDay</td>
<td>Int8</td>
<td>2</td>
<td>The first day of the week (Sunday or Monday). Days of the week are numbered from 0 to 6 starting with Sunday = 0.</td>
</tr>
<tr>
<td>prefTimeFormat</td>
<td>TimeFormatType</td>
<td>2</td>
<td>The format used to display time values.</td>
</tr>
<tr>
<td>prefNumberFormat</td>
<td>NumberFormatType</td>
<td>2</td>
<td>The format used for numbers, with regards to the thousands separator and the decimal point.</td>
</tr>
<tr>
<td>prefAutoOffDuration</td>
<td>UInt8</td>
<td>2</td>
<td>Minutes of user idle time before the device powers off. The default value for this preference is specified by the defaultAutoOffDuration constant.</td>
</tr>
</tbody>
</table>
### Preferences

#### Preferences Data Types

**prefAutoOffDuration** is replaced by **prefAutoOffDurationSecs** in version 8 of the preferences structure.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefSysSound LevelV20</td>
<td><code>SoundLevelTypeV20</code></td>
<td>2</td>
<td>Specifies whether system sounds are enabled or disabled.</td>
</tr>
<tr>
<td>prefGameSound LevelV20</td>
<td><code>SoundLevelTypeV20</code></td>
<td>2</td>
<td>Specifies whether game sound effects are on or off.</td>
</tr>
<tr>
<td>prefAlarmSound LevelV20</td>
<td><code>SoundLevelTypeV20</code></td>
<td>2</td>
<td>Specifies whether sound alarms are on or off.</td>
</tr>
<tr>
<td>prefHidePrivate RecordsV33</td>
<td>Boolean</td>
<td>2</td>
<td>If true, applications should not display database records that have the secret attribute bit set.</td>
</tr>
<tr>
<td>prefDeviceLocked</td>
<td>Boolean</td>
<td>2</td>
<td>If true, the device is locked. When the device is locked, it remains so until the user enters the password.</td>
</tr>
<tr>
<td>prefLocal SyncRequires Password</td>
<td>Boolean</td>
<td>2</td>
<td>If true, the user must enter a password before a HotSync® operation can be performed.</td>
</tr>
<tr>
<td>prefRemote SyncRequires Password</td>
<td>Boolean</td>
<td>2</td>
<td>If true, the user must enter a password on the desktop computer before a HotSync operation can be performed.</td>
</tr>
<tr>
<td>prefSysBattery Kind</td>
<td><code>SysBatteryKind</code></td>
<td>2</td>
<td>The type of batteries installed. Use <code>SysBatteryInfo</code> to retrieve the battery type instead of this preference.</td>
</tr>
</tbody>
</table>

---

*Table 43.1 SystemPreferencesChoice Constants (continued)*
<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefMinutes WestOfGMT</td>
<td>UInt32</td>
<td>2</td>
<td>The time zone given as minutes <strong>east</strong> of Greenwich Mean Time (GMT). For preferences version 9 and higher, use prefTimeZone instead.</td>
</tr>
<tr>
<td>prefDaylight Savings</td>
<td>DaylightSaving STypes</td>
<td>2</td>
<td>The type of daylight savings correction. For preferences version 9 and higher, use prefDaylightSavingAdjustment instead.</td>
</tr>
<tr>
<td>prefRonamatic Char</td>
<td>UInt16</td>
<td>2</td>
<td>The virtual character generated when the user enters the ronamatic stroke. The ronamatic stroke is dragging the pen from the Graffiti® area to the top of the screen.</td>
</tr>
<tr>
<td>prefHard1Char AppCreator</td>
<td>UInt32</td>
<td>2</td>
<td>The creator ID of the application to be launched by the left-most hard key (the Date Book button by default).</td>
</tr>
<tr>
<td>prefHard2Char AppCreator</td>
<td>UInt32</td>
<td>2</td>
<td>The creator ID of the application to be launched by the second hard key from the left (the Address button by default).</td>
</tr>
<tr>
<td>prefHard3Char AppCreator</td>
<td>UInt32</td>
<td>2</td>
<td>The creator ID of the application to be launched by the second hard key from the right (the To Do List button by default).</td>
</tr>
</tbody>
</table>
### Table 43.1 SystemPreferencesChoice Constants (continued)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefHard4Char</td>
<td>AppCreator</td>
<td>UInt32</td>
<td>2 The creator ID of the application to be launched by the right-most hard key (the Memo Pad button by default).</td>
</tr>
<tr>
<td>prefCalcChar</td>
<td>AppCreator</td>
<td>UInt32</td>
<td>2 The creator ID of the application to be launched by the Calculator silkscreen button.</td>
</tr>
<tr>
<td>prefHardCradle</td>
<td>CharAppCreator</td>
<td>UInt32</td>
<td>2 The creator ID of the application to be launched by the hard key on the HotSync cradle.</td>
</tr>
<tr>
<td>prefLauncher</td>
<td>AppCreator</td>
<td>UInt32</td>
<td>2 The creator ID of the application to be launched by the Applications silkscreen button.</td>
</tr>
<tr>
<td>prefHardCradle2</td>
<td>CharAppCreator</td>
<td>UInt32</td>
<td>2 The creator ID of the application to be launched by the HotSync button on the modem.</td>
</tr>
<tr>
<td>prefAnimation</td>
<td>AnimationLevel</td>
<td>2</td>
<td>The sound level for system sounds, such as taps and beeps. This is a value from 0 to sndMaxAmp.</td>
</tr>
<tr>
<td>prefSysSoundVolume</td>
<td>UInt16</td>
<td>3</td>
<td>The sound level for game sounds. This is a value from 0 to sndMaxAmp.</td>
</tr>
<tr>
<td>prefGameSoundVolume</td>
<td>UInt16</td>
<td>3</td>
<td>The sound level for game sounds. This is a value from 0 to sndMaxAmp.</td>
</tr>
</tbody>
</table>
### Preferences

**Preferences Data Types**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefAlarm</td>
<td>UInt16</td>
<td>3</td>
<td>The sound level for alarms. This is a value from 0 to sndMaxAmp.</td>
</tr>
<tr>
<td>SoundVolume</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prefBeamReceive</td>
<td>Boolean</td>
<td>3</td>
<td>If true, the device can receive beams from other devices. If false, the device cannot receive beams but can still send them.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prefCalibrate</td>
<td>Boolean</td>
<td>3</td>
<td>If true, the user must recalibrate the digitizer after a soft reset. The default is false.</td>
</tr>
<tr>
<td>DigitizerAtReset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prefSystem</td>
<td>UInt16</td>
<td>4</td>
<td>The resource ID of the keyboard panel.</td>
</tr>
<tr>
<td>KeyboardID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prefDefSerial</td>
<td>UInt32</td>
<td>4</td>
<td>The creator ID of the default serial plug-in database.</td>
</tr>
<tr>
<td>PlugIn</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prefStayOn</td>
<td>Boolean</td>
<td>5</td>
<td>If true, the device stays powered on when it is in the cradle.</td>
</tr>
<tr>
<td>WhenPluggedIn</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prefStayLit</td>
<td>Boolean</td>
<td>5</td>
<td>If true and prefStayOnWhenPluggedIn is true, the device stays lit when it is in its cradle.</td>
</tr>
<tr>
<td>WhenPluggedIn</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prefAntenna</td>
<td>UInt32</td>
<td>6</td>
<td>The creator ID of the application to launch when the antenna is raised (used only for devices with built-in antennas).</td>
</tr>
<tr>
<td>CharAppCreator</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

834  *Palm OS Programmer’s API Reference*
### Table 43.1 SystemPreferencesChoice Constants (continued)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefMeasurement System</td>
<td>MeasurementSystemType</td>
<td>7</td>
<td>The system of measurement to use.</td>
</tr>
<tr>
<td>prefShow Private Records</td>
<td>privateRecordViewEnum</td>
<td>8</td>
<td>Specifies whether the private records should be displayed, masked, or completely hidden.</td>
</tr>
<tr>
<td>prefAutoOff DurationSecs</td>
<td>UInt16</td>
<td>8</td>
<td>Seconds of user idle time before the device powers off. The default value for this preference is specified by the defaultAutoOffDurationSecs constant.</td>
</tr>
<tr>
<td>prefTimeZone</td>
<td>Int16</td>
<td>9</td>
<td>The time zone given as minutes east of Greenwich Mean Time (GMT).</td>
</tr>
<tr>
<td><strong>IMPORTANT:</strong> Changing the value of this preference does not update the current time.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prefDaylight Saving Adjustment</td>
<td>Int16</td>
<td>9</td>
<td>The number of minutes to add to the current time for daylight savings time.</td>
</tr>
<tr>
<td><strong>IMPORTANT:</strong> Changing the value of this preference does not update the current time.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prefTimeZone Country</td>
<td>CountryType</td>
<td>9</td>
<td>The country selected to specify what the time zone is.</td>
</tr>
</tbody>
</table>
prefAutoLockType  SecurityAutoLockType  9  Specifies when the auto-locking feature should take effect. Possibilities are upon power off, at a preset time, or after a certain number of seconds.

prefAutoLockTime  UInt32  9  The time value for the auto-locking feature if the system should lock itself after a delay or at a preset time. Depending on the value of prefAutoLockType, this value is either an absolute date and time given as the number of seconds since January 1, 1904 or a timeout value given as a number of seconds from the current time.

prefAutoLockTimeFlag  Boolean  9  If true, prefAutoLockTime is given in minutes. If false, the time is given in hours.

prefLanguage  LanguageType  9  The language that the device should use.

prefAttentionFlags  AttnFlagsType  9  The user’s preferences for receiving attention signals. The returned value is a bit mask that should be tested (using the & operator) with one of the following values:

Table 43.1 SystemPreferencesChoice Constants (continued)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefAutoLockType</td>
<td>SecurityAutoLo\ckType</td>
<td></td>
<td>Specifies when the auto-locking feature should take effect. Possibilities are upon power off, at a preset time, or after a certain number of seconds.</td>
</tr>
<tr>
<td>prefAutoLockTime</td>
<td>UInt32</td>
<td></td>
<td>The time value for the auto-locking feature if the system should lock itself after a delay or at a preset time. Depending on the value of prefAutoLockType, this value is either an absolute date and time given as the number of seconds since January 1, 1904 or a timeout value given as a number of seconds from the current time.</td>
</tr>
<tr>
<td>prefAutoLockTimeFlag</td>
<td>Boolean</td>
<td></td>
<td>If true, prefAutoLockTime is given in minutes. If false, the time is given in hours.</td>
</tr>
<tr>
<td>prefLanguage</td>
<td>LanguageType</td>
<td></td>
<td>The language that the device should use.</td>
</tr>
<tr>
<td>prefAttentionFlags</td>
<td>AttnFlagsType</td>
<td></td>
<td>The user’s preferences for receiving attention signals. The returned value is a bit mask that should be tested (using the &amp; operator) with one of the following values:</td>
</tr>
</tbody>
</table>
Preferences
Preferences Data Types

Table 43.1 SystemPreferencesChoice Constants (continued)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefDefaultAppCreator</td>
<td>UInt32</td>
<td>9</td>
<td>Creator ID of the application that is launched after a reset. If 0, the system default application is launched.</td>
</tr>
<tr>
<td>prefLocale</td>
<td>LmLocaleType</td>
<td>9</td>
<td>The device’s current locale, which specifies the country and language.</td>
</tr>
<tr>
<td>prefDefFepPlugInCreator</td>
<td>UInt32</td>
<td>10</td>
<td>Creator ID of the default FEP plug-in.</td>
</tr>
<tr>
<td>prefColorThemeID</td>
<td>DmResID</td>
<td>10</td>
<td>Resource ID of the color theme.</td>
</tr>
</tbody>
</table>

Note that you can override the values in prefAttentionFlags when you make Attention Manager function calls. See the section “Getting the User’s Attention” on page 283 of the Palm OS Programmer’s Companion, vol. I for more information.

Preferences in the User Interface

Table 43.2 shows the SystemPreferencesChoice constants and how they correspond to the values that users can set in the Preferences and Security applications. For further information about each preference, see Table 43.1.
### Table 43.2 Preferences set in standard apps

<table>
<thead>
<tr>
<th>Application/Panel</th>
<th>Field</th>
<th>SystemPreferencesChoice Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preferences</td>
<td>Auto-off After</td>
<td>prefAutoOffDuration, prefAutoOffDurationSecs</td>
</tr>
<tr>
<td>General panel</td>
<td></td>
<td>(Palm OS 3.5 and higher)</td>
</tr>
<tr>
<td></td>
<td>Stay on in Cradle</td>
<td>prefStayOnWhenPluggedIn</td>
</tr>
<tr>
<td></td>
<td>System Sound</td>
<td>prefSysSoundLevelV20, prefSysSoundVolume</td>
</tr>
<tr>
<td></td>
<td>Alarm Sound</td>
<td>prefAlarmSoundLevelV20, prefAlarmSoundVolume</td>
</tr>
<tr>
<td></td>
<td>Alarm Vibrate¹</td>
<td>prefAttentionFlags</td>
</tr>
<tr>
<td></td>
<td>Alarm LED²</td>
<td>prefAttentionFlags</td>
</tr>
<tr>
<td></td>
<td>Game Sound</td>
<td>prefGameSoundLevelV20, prefGameSoundVolume</td>
</tr>
<tr>
<td></td>
<td>Beam Receive field</td>
<td>prefBeamReceive</td>
</tr>
<tr>
<td>Preferences</td>
<td>Set Time Zone field</td>
<td>prefTimeZone</td>
</tr>
<tr>
<td>Date &amp; Time panel</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Daylight Saving</td>
<td>prefDaylightSaving Adjustment</td>
</tr>
<tr>
<td>Preferences</td>
<td>Preset to</td>
<td>prefCountry</td>
</tr>
<tr>
<td>Formats panel</td>
<td>Time</td>
<td>prefTimeFormat</td>
</tr>
<tr>
<td></td>
<td>Date</td>
<td>prefDateFormat, prefLongDateFormat</td>
</tr>
<tr>
<td></td>
<td>Week starts</td>
<td>prefWeekStartDay</td>
</tr>
<tr>
<td></td>
<td>Numbers</td>
<td>prefNumberFormat</td>
</tr>
</tbody>
</table>
Table 43.2 Preferences set in standard apps *(continued)*

<table>
<thead>
<tr>
<th>Application/Panel</th>
<th>Field</th>
<th>SystemPreferencesChoice Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preferences application Buttons panel</td>
<td>Buttons on main panel</td>
<td>prefHard1CharAppCreator, prefHard2CharAppCreator, prefHard3CharAppCreator, prefHard4CharAppCreator, prefCalcCharAppCreator, prefLauncherAppCreator</td>
</tr>
<tr>
<td>Pen button</td>
<td></td>
<td>prefRonamaticChar</td>
</tr>
<tr>
<td>HotSync button</td>
<td></td>
<td>prefHardCradleCharAppCreator, prefHardCradle2CharAppCreator</td>
</tr>
<tr>
<td>Security application</td>
<td>Current Privacy</td>
<td>prefHidePrivateRecordsV33, prefShowPrivateRecords</td>
</tr>
<tr>
<td>Security application</td>
<td>Lock &amp; Turn Off button</td>
<td>prefDeviceLocked, prefAutoLockType, prefAutoLockTime, prefAutoLockTimeFlag</td>
</tr>
</tbody>
</table>

1. The Alarm Vibrate and Alarm LED preferences only appear on handhelds running Palm OS 4.0 and later that have the appropriate hardware capabilities. If you install Palm OS 4.0 on an older device, these preferences do not display.

Preferences Constants

The following table describes the constants defined in the Preferences.h header file.
<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultAutoLockTime</td>
<td>0</td>
<td>Initial setting for the prefAutoLockTime preference.</td>
</tr>
<tr>
<td>defaultAutoLockTimeFlag</td>
<td>0</td>
<td>Initial setting for the prefAutoLockTimeFlag preference.</td>
</tr>
<tr>
<td>defaultAutoLockType</td>
<td>never</td>
<td>Initial setting for the prefAutoLockType preference.</td>
</tr>
<tr>
<td>defaultAutoOffDuration</td>
<td>2</td>
<td>Initial setting for the prefAutoOffDuration preference, given in minutes.</td>
</tr>
<tr>
<td>defaultAutoOffDurationSecs</td>
<td>120</td>
<td>Initial setting for the prefAutoOffDurationSecs preference, given in seconds. The value is 2 times the number of seconds in a minute, or 120 seconds.</td>
</tr>
<tr>
<td>defaultAlarmSoundLevel</td>
<td>s1On</td>
<td>Initial setting for the prefAlarmSoundLevelV20 preference.</td>
</tr>
<tr>
<td>defaultAlarmSoundVolume</td>
<td>sndMaxAmp</td>
<td>Initial setting for the prefAlarmSoundVolume preference.</td>
</tr>
<tr>
<td>defaultGameSoundLevel</td>
<td>s1On</td>
<td>Initial setting for the prefGameSoundLevelV20 preference.</td>
</tr>
<tr>
<td>defaultGameSoundVolume</td>
<td>sndMaxAmp</td>
<td>Initial setting for the prefGameSoundVolume preference.</td>
</tr>
<tr>
<td>defaultSysSoundLevel</td>
<td>s1On</td>
<td>Initial setting for the prefSysSoundLevelV20 preference.</td>
</tr>
<tr>
<td>defaultSysSoundVolume</td>
<td>sndMaxAmp</td>
<td>Initial setting for the prefSysSoundVolume preference.</td>
</tr>
</tbody>
</table>
Preferences
Preferences Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noPreferenceFound</td>
<td>-1</td>
<td>The value that PrefGetAppPreferences returns to indicate that preferences couldn’t be found.</td>
</tr>
<tr>
<td>preferenceDataVer2</td>
<td>2</td>
<td>Version 2 of the system preferences structure, used for Palm OS 2.0.</td>
</tr>
<tr>
<td>preferenceDataVer3</td>
<td>3</td>
<td>Version 3 of the system preferences structure, used for Palm OS 3.0.</td>
</tr>
<tr>
<td>preferenceDataVer4</td>
<td>4</td>
<td>Version 4 of the system preferences structure, used for Palm OS 3.1.</td>
</tr>
<tr>
<td>preferenceDataVer5</td>
<td>5</td>
<td>Version 5 of the system preferences structure, used for Palm OS 3.2.</td>
</tr>
<tr>
<td>preferenceDataVer6</td>
<td>6</td>
<td>Version 6 of the system preferences structure, used for Palm OS 3.3.</td>
</tr>
<tr>
<td>preferenceDataVer8</td>
<td>8</td>
<td>Version 8 of the system preferences structure, used for Palm OS 3.5.</td>
</tr>
<tr>
<td>preferenceDataVer9</td>
<td>9</td>
<td>Version 9 of the system preferences structure, used for Palm OS 4.0.</td>
</tr>
<tr>
<td>preferenceDataVer10</td>
<td>10</td>
<td>Version 10 of the system preferences structure, used for Palm OS 5.0.</td>
</tr>
<tr>
<td>preferenceDataVer11</td>
<td>11</td>
<td>Version 11 of the system preferences structure, used for Palm OS 5.1.</td>
</tr>
<tr>
<td>preferenceDataVerLatest</td>
<td>preferenceDataVer11</td>
<td>The latest version of the system preferences structure.</td>
</tr>
</tbody>
</table>
Preferences Functions

PrefGetAppPreferences

**Purpose**
Return a copy of an application’s preferences resource.

**Declared In**
Preferences.h

**Prototype**
```c
Int16 PrefGetAppPreferences (UInt32 creator,
   UInt16 id, void *prefs, UInt16 *prefsSize,
   Boolean saved)
```

**Parameters**
- `creator` Creator ID of the application that owns the preferences.
- `id` ID number of the preferences resource to retrieve. The IDs 0x8000 through 0xFFFF are reserved for system use.
- `prefs` Pointer to a buffer to hold the preferences.
- `prefsSize` Pointer to the size of the `prefs` buffer passed in. Upon return, contains the number of bytes actually written or the number of bytes needed for the `prefs` structure.

To determine the required size for the `prefs` structure, set `prefsSize` to 0 and pass NULL for `prefs`. Upon return, the `prefsSize` parameter contains the required size. Never set `prefs` to NULL without also setting `prefsSize` to 0.

Always compare the value returned in this parameter with the value you passed in. If the two values differ, you need to resize the `prefs` structure and call this function again.
-> saved

If true, retrieve the preferences from the “saved” preferences database, which is backed up during a HotSync operation. If false, retrieve the preferences from the “unsaved” preferences database, which is usually not backed up during a HotSync operation.

Result

Returns the version number of the retrieved preferences resource, or returns the constant noPreferenceFound if the preferences resource wasn’t found. The returned version number is the same version number that was passed to the PrefSetAppPreferences function.

Comments

Use this function to retrieve the preferences that you previously set with the PrefSetAppPreferences function. You typically call this function in your StartApplication function upon a normal launch. The values of the id and saved parameters should be the same as you specified when calling PrefSetAppPreferences, and the prefs parameter should be a structure of the same type as you passed to PrefSetAppPreferences. Most applications store all preferences in a single preferences resource retrieved by a single call to PrefGetAppPreferences, but this is not required. You can use multiple preferences resources if you wish.

Applications typically call PrefGetAppPreferences twice: once with prefs set to NULL and prefsSize set to 0 to determine how much memory to allocate for the preferences structure, and a second time after allocating the required amount of memory to obtain a copy of the application’s preferences resource.

The version number returned by this function allows you to handle the case where a new version of the application is being run for the first time. You can compare the value returned by this function with the current version number to determine if you need to set default values for any preferences created by the current release. For more information, see the section “Updating Preferences Upon a New Release” on page 329 of the Palm OS Programmer’s Companion, vol. I.

Compatibility

Implemented only if 2.0 New Feature Set is present.

See Also

PrefSetPreferences, PrefGetAppPreferencesV10
**PrefGetAppPreferencesV10**

**Purpose**  
Return a copy of an application’s preferences.

**Declared In**  
Preferences.h

**Prototype**  
Boolean PrefGetAppPreferencesV10 (UInt32 type, Int16 version, void *prefs, UInt16 prefsSize)

**Parameters**
- **type**  
Creator ID of the application that owns the preferences.
- **version**  
Version number of the application’s preferences.
- **prefs**  
Pointer to a buffer to hold preferences.
- **prefsSize**  
Size of the buffer passed.

**Result**  
Returns false if the preference resource was not found or the preference resource contains the wrong version number.

**Comments**  
The content and format of an application preference is application-dependent.

**Compatibility**  
This function corresponds to the 1.0 version of PrefGetAppPreferences.

**See Also**  
PrefSetPreferences, PrefGetAppPreferences
**PrefGetPreference**

**Purpose**
Return a system preference. Use this function instead of **PrefGetPreferences**.

**Declared In**
Preferences.h

**Prototype**
UInt32 PrefGetPreference(SystemPreferencesChoice choice)

**Parameters**
- choice
  A constant that specifies what preference to retrieve. See **SystemPreferencesChoice**.

**Result**
Returns the system preference or 0 if the preference could not be found. On debug ROMs, a non-fatal error message is also displayed if the specified preference cannot be found.

**Comments**
This function replaces the 1.0 function **PrefGetPreferences**. While **PrefGetPreferences** only lets you retrieve the whole system preferences structure, this function lets you specify which preference to retrieve.

**Compatibility**
Implemented only if **2.0 New Feature Set** is present.

**See Also**
PrefSetPreference, PrefGetAppPreferences, PrefGetAppPreferencesV10

**PrefGetPreferences**

**Purpose**
Return a copy of the system preferences.

**Declared In**
Preferences.h

**Prototype**
void PrefGetPreferences(SystemPreferencesPtr p)

**Parameters**
- p
  Pointer to system preferences.

**Result**
Returns nothing. Stores the system preferences in p.
Comments
The p parameter points to a memory block allocated by the caller that is filled in by this function.
This function is often called in StartApplication to get localized settings.

NOTE: This function can only be used to retrieve preferences that were in the 1.0 version of the preferences structure.

See Also  PrefSetPreferences

PrefOpenPreferenceDB

Purpose
Return a handle to either the saved or unsaved preference database.

Declared In
Preferences.h

Prototype
DmOpenRef PrefOpenPreferenceDB (Boolean saved)

Parameters
-> saved
If true, open the “saved” preferences database, which is backed up during a HotSync operation. If false, open the “unsaved” preferences database, which usually is not backed up during a HotSync operation.

Result
Returns the handle, or 0 if an error results.

Compatibility
Implemented only if 2.0 New Feature Set is present.

See Also
PrefGetPreference, PrefSetPreference, PrefOpenPreferenceDBV10
**PrefOpenPreferenceDBV10**

**Purpose**
Return a handle to the system preference database.

**Declared In**
Preferences.h

**Prototype**
DmOpenRef PrefOpenPreferenceDBV10 (void)

**Parameters**
None.

**Result**
Returns the handle, or 0 if an error results.

**Compatibility**
This function corresponds to the 1.0 version of PrefOpenPreferenceDB.

**See Also**
PrefGetPreferences, PrefSetPreferences

**PrefSetAppPreferences**

**Purpose**
Set an application’s preferences in the specified preferences database.

**Declared In**
Preferences.h

**Prototype**
void PrefSetAppPreferences (UInt32 creator, UInt16 id, Int16 version, const void *prefs, UInt16 prefsSize, Boolean saved)

**Parameters**
- **creator**
  Creator ID of the application that owns this preference.

- **id**
  ID number of the preference to set. An application can have multiple preferences. The IDs 0x8000 through 0xFFFF are reserved for system use.

- **version**
  Version number of the application’s preferences.
Preferences
Preferences Functions

-> prefs     Pointer to a buffer that holds the current value of the preferences structure. Pass NULL if you want to delete the preferences.

-> prefsSize Size of the buffer passed. Pass 0 if you want to delete the preferences structure.

-> saved    If true, saves the preferences in the “saved” preferences database. If not, saves the preferences in the “unsaved” preferences database.

Result    Returns nothing.

Comments You typically call this function in your StopApplication function to save the current state of the application or if the user has changed an application preference during the current session.

The “saved” preferences database is backed up when a user performs the HotSync operation. The “unsaved” preferences database is not backed up by default. (The user can use a third-party tool to set the backup bit in the “unsaved” preferences database, which would cause it to be backed up.) Both the “saved” and the “unsaved” preferences reside in the storage heap and thus persist across soft resets. The only way that preferences are lost is if a hard reset is performed. “Which Preferences Database to Use” on page 327 of the Palm OS Programmer’s Companion, vol. I describes how to choose between the “saved” and “unsaved” preferences databases.

The version number that you pass as the version parameter is returned when the preferences are retrieved by PrefGetAppPreferences. You can use this version number to determine if a new release of the application is being run for the first time. For more information, see the section “Updating Preferences Upon a New Release” on page 329 of the Palm OS Programmer’s Companion, vol. I.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also PrefSetAppPreferencesV10
PrefSetAppPreferencesV10

Purpose  Save an application’s preferences in the preferences database.

Declared In  Preferences.h

Prototype  void PrefSetAppPreferencesV10 (UInt32 creator, Int16 version, void *prefs, UInt16 prefsSize)

Parameters  
- creator  Creator ID of the application that owns this preference.
- version  Version number of the application.
- prefs  Pointer to a buffer holding preferences.
- prefsSize  Size of the buffer passed.

Result  Returns nothing.

Comments  The content and format of an application preference is application-dependent.

Compatibility  This function corresponds to the 1.0 version of PrefSetAppPreferences.

See Also  PrefSetAppPreferences, PrefGetPreferences
### PrefSetPreference

**Purpose**
Set a system preference. Using this function instead of `PrefSetPreferences` allows you to set selected preferences without having to access the whole structure.

**Declared In**
Preferences.h

**Prototype**
```c
void PrefSetPreference
    (SystemPreferencesChoice choice, UInt32 value)
```

**Parameters**
- `-> choice` A `SystemPreferencesChoice` constant specifying the preference to be set.
- `-> value` Value to assign to the preference.

**Result**
Returns nothing. If the specified preference cannot be found, displays a non-fatal error message on debug ROMs. On release ROMs, this function fails silently.

**Compatibility**
Implemented only if [2.0 New Feature Set](#) is present.

### PrefSetPreferences

**Purpose**
Set the system preferences.

**Declared In**
Preferences.h

**Prototype**
```c
void PrefSetPreferences (SystemPreferencesPtr p)
```

**Parameters**
- `-> p` Pointer to system preferences.

**Result**
Returns nothing.

**Comments**
Unless there’s a reason for you to access the whole preferences structure, call `PrefSetPreference` instead.
NOTE: This function can only be used to set preferences that were in the 1.0 version of the preferences structure.

See Also  PrefGetPreferences
Preferences

Preferences Functions
Rectangles

This chapter provides reference material for the rectangles API, declared in the header file `Rect.h`. It is divided into the following sections:

- Rectangle Data Structures
- Rectangle Functions

Rectangle Data Structures

**PointType**

The `PointType` structure defines a point within a window or on the screen.

```c
typedef struct PointType {
    Coord x;
    Coord y;
} PointType;
```

**Field Descriptions**

- `x` Horizontal coordinate.
- `y` Vertical coordinate.

**RectangleType**

The `RectangleType` structure defines a rectangular portion of a window or of the screen.

```c
typedef struct RectangleType {
    PointType topLeft;
    PointType extent;
} RectangleType;

typedef RectangleType *RectanglePtr;
```
Field Descriptions

- **topLeft**: Coordinates of the upper-left corner of the rectangle relative to the window or screen in which the rectangle resides.
- **extent**: Width (extent.x) and height (extent.y) of the rectangle.

Rectangle Functions

**RctCopyRectangle**

**Purpose**: Copy the source rectangle to the destination rectangle.

**Declared In**: Rect.h

**Prototype**:
```c
void RctCopyRectangle
  (const RectangleType* srcRectP,
   RectangleType* dstRectP)
```

**Parameters**:
- `srcRectP`: A pointer to the rectangle to be copied.
- `dstRectP`: A pointer to the destination rectangle.

**See Also**: RctSetRectangle

**RctGetIntersection**

**Purpose**: Determine the intersection of two rectangles.

**Declared In**: Rect.h

**Prototype**:
```c
void RctGetIntersection
  (const RectangleType* r1P,
   const RectangleType* r2P, RectangleType* r3P)
```

**Parameters**:
- `r1P`: A pointer to a source rectangle.
r2P A pointer to the other source rectangle.

r3P Upon return, points to a rectangle representing the intersection of r1 and r2.

Comments The rectangle type `RectangleType`, which is pointed to by `RectanglePtr`, stores the coordinates for the top-left corner of the rectangle plus the rectangle’s width and height. This function returns in the `r3` parameter a pointer to the rectangle that represents the intersection of the first two rectangles.

If the rectangles r1 and r2 do not intersect, r3 contains a rectangle whose top-left coordinate is the maximum of r1 and r2’s top-left coordinates and whose extent varies based on the location of the two rectangles.

Compatibility On releases prior to Palm OS® 3.5, if rectangles r1 and r2 don’t intersect, r3 contains a rectangle that begins at coordinates (0,0) and has 0 width and 0 height. On Palm OS 3.5 and later, if the two rectangles don’t intersect then r3 contains a rectangle in which one or both of the extent coordinates is zero.

RctInsetRectangle

Purpose Move all of the boundaries of a rectangle by a specified offset.

Declared In Rect.h

Prototype `void RctInsetRectangle (RectangleType* rP, Coord insetAmt)`

Parameters rP A pointer to the rectangle.

insetAmt Number of pixels to move the boundaries. This can be a negative number.

Comments The rectangle type `RectangleType`, which is pointed to by `RectanglePtr`, stores the coordinates for the top-left corner of the rectangle plus the rectangle’s width and height. This function adds `insetAmt` to the x and y values of the top-left coordinate and then
adjusts the width and the height accordingly so that all of the sides of the rectangle are contracted or expanded by the same amount.

A positive `insetAmt` creates a smaller rectangle that is contained inside the old rectangle’s boundaries. A negative `insetAmt` creates a larger rectangle that surrounds the old rectangle.

**See Also**  
[RctOffsetRectangle](#)

### RctOffsetRectangle

**Purpose**  
Move the top and left boundaries of a rectangle by the specified values.

**Declared In**  
Rect.h

**Prototype**  
```c
void RctOffsetRectangle (RectangleType* rP,  
Coord deltaX, Coord deltaY)
```

**Parameters**  
- `rP`: A pointer to the rectangle.
- `deltaX`: Number of pixels to move the left boundary. This can be a negative number.
- `deltaY`: Number of pixels to move the top boundary. This can be a negative number.

**Comments**  
The rectangle type `RectangleType`, which is pointed to by RectanglePtr, stores the coordinates for the top-left corner of the rectangle plus the rectangle’s width and height. This function adds `deltaX` to the x value of the top-left coordinate and `deltaY` to the y value. The width and height are unchanged. Thus, this function shifts the position of the rectangle by the `deltaX` and `deltaY` amounts.

**See Also**  
[RctInsetRectangle](#)
### RctPtInRectangle

**Purpose** Determine if a point lies within a rectangle’s boundaries.

**Declared In** Rect.h

**Prototype**

```c
Boolean RctPtInRectangle (Coord x, Coord y, const RectangleType* rP)
```

**Parameters**

- **x**
  The x coordinate of the point.
- **y**
  The y coordinate of the point.
- **rP**
  The rectangle.

**Result**

Returns `true` if the point \((x, y)\) lies within the boundaries of rectangle \(r\), `false` otherwise.

### RctSetRectangle

**Purpose** Sets a rectangle’s values.

**Declared In** Rect.h

**Prototype**

```c
void RctSetRectangle (RectangleType* rP, Coord left, Coord top, Coord width, Coord height)
```

**Parameters**

- **rP**
  A pointer to the rectangle to be set.
- **left**
  The x value for the top-left coordinate of the rectangle.
- **top**
  The y value for the top-left coordinate of the rectangle.
- **width**
  The rectangle’s width.
- **height**
  The rectangle’s height.

**See Also** RctCopyRectangle
Sound Manager

This chapter describes the API that’s defined by the Sound Manager. You use this API to make and record sounds.

The Sound Manager controls two independent sound facilities:

- Single voice, monophonic, square-wave sound synthesis, useful for system beeps. This is the traditional (pre-OS 5) PalmSource sound; let’s call it “simple sound.”
- Stereo, multi-format, sampled data recording and playback (new in Palm OS 5). We’ll call it “sampled sound.”

The API for these two facilities are entirely separate; the simple sound API is described first...

- Simple Sound Structures and Constants
- Simple Sound Functions
- Simple Sound Application-Defined Functions

...followed by the sampled sound API.

- Sampled Sound Structures, Constants, and Data Types
- Sampled Sound Functions
- Sampled Sound Application-Defined Functions

All elements described here are declared in SoundMgr.h.

Overview

There are three ways to play a simple sound:

- You can play a single tone of a given pitch, amplitude, and duration by calling SndDoCmd.
- You can play a pre-defined system sound through SndPlaySystemSound.
- You can play a tune by passing in a Level 0 Standard MIDI file (SMF) through the SndPlaySmf function.
There are a handful of other simple sound functions, but they mostly support these three fundamental functions.

Over in the sampled sound facilities, there are two fundamental functions:

- **SndStreamCreate** opens a new sampled sound “stream” from/into which you record/playback buffers of “raw” data. The trick is that you first have to configure the stream to tell it how to interpret the data.

- **SndPlayResource** is used to playback sound data that’s read from a (formatted) sound file. The function configures the playback stream for you, based on the format information in the sound file header. Currently, only uncompressed WAV and IMA ADPCM WAV formats are recognized (this also known as DVI ADPCM). Note that SndPlayResource is only used to play back sound; it can’t be used for recording.

The Sound Manager also provides functions that let you set the volume and stereo panning for individual recording and playback streams. See SndStreamSetVolume and SndStreamSetPan.

For more information on the Sound Manager, see the section “Sound” in the Palm OS Programmer’s Companion, vol. I.

---

**Simple Sound Structures and Constants**

**SndCallbackInfoType**

Structure that encapsulates a callback function and its argument data. SndCallbackInfoType is used by the SndSmfCallbacksType structure, which is used to list the callback functions that are called during SMF playback.

```c
typedef struct SndCallbackInfoType {
    MemPtr    funcP;
    UInt32    dwUserData;
} SndCallbackInfoType;
```
The fields are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>funcP</td>
<td>A pointer to the callback function.</td>
</tr>
<tr>
<td>dwUserData</td>
<td>Data that’s passed as an argument to the function.</td>
</tr>
</tbody>
</table>

**SndCmdIDTag**

The SndCmdIDTag enumeration contains constants that represent specific sound operations used in simple sound playback.

```c
typedef enum SndCmdIDTag {
    sndCmdFreqDurationAmp = 1,
    sndCmdNoteOn,
    sndCmdFrqOn,
    sndCmdQuiet
};
```

See [SndDoCmd](#) for descriptions of the operations that are represented by these values.

**SndCommandType**

The SndCommandType structure encapsulates a sound synthesis operation and its associated parameters. Used by the SndDoCmd function.

```c
typedef struct SndCommandType {
    SndCmdIDType   cmd;
    UInt8          reserved;
    Int32          param1;
    UInt16         param2;
    UInt16         param3;
} SndCommandType;
```
The fields are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
| cmd       | Constant that represents a sound operation. The operations are listed and described in `SndDoCmd`.
| reserved  | Don’t touch this one. |
| param1, param2, param3 | Operation-specific parameters. The parameters’ meanings are described in `SndDoCmd`.

**sndMaxAmp**

Constant that defines the upper limit for simple sound amplitude values. Use this value with the simple sound API only (such as `SndDoCmd`). The `sndMaxAmp` value is *not* compatible with the sampled sound amplitude range.

```c
#define sndMaxAmp 64
```

**SndMidiListItemType**

Structure that locates a MIDI file. The structure is used by the `SndCreateMidiList` function.

```c
typedef struct SndMidiListItemType {
  Char  name[sndMidiNameLength];
  UInt32  uniqueRecID;
  MemHandle  dbH;
} SndMidiListItemType;
```

The fields are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The null-terminated name of the MIDI file.</td>
</tr>
<tr>
<td>uniqueRecID</td>
<td>The ID of the record that holds the MIDI file.</td>
</tr>
<tr>
<td>dbH</td>
<td>Handle to the database that holds the record.</td>
</tr>
</tbody>
</table>
### sndMidiNameLength

Constant that defines the maximum string length (including the null terminator) for a MIDI file or MIDI track name.

```c
#define sndMidiNameLength 32
```

### SndMidiRecHdrType

Structure that encapsulates the header of a MIDI record.

```c
typedef struct SndMidiRecHdrType {
  UInt32  signature;
  UInt8   bDataOffset;
  UInt8   reserved;
} SndMidiRecHdrType;
```

The fields are:

- **signature**: Always set to `sndMidiRecSignature`.
- **bDataOffset**: Offset, in bytes, from the beginning of the record to the first byte of the MIDI data. Always set to 0.
- **reserved**: Always set to 0.

**Figure 45.1** depicts a complete Palm OS® MIDI record.

---

To get to the track name, you have to bump the structure pointer yourself. For example:
Char *trackName = (Char *)myMidiRecHdr + sizeof(SndMidiRecHdrType);

The MIDI track name is null-terminated, even if it’s empty. It’s at least one byte long and at most \texttt{sndMidiNameLength} bytes long.

\textbf{sndMidiRecSignature}

Endian-cognizant constant that’s used to tag a MIDI record. The constant is used as the value of the signature field of the \texttt{SndMidiRecHdrType} structure.

\begin{verbatim}
#if CPU_ENDIAN == CPU_ENDIAN_BIG
#define sndMidiRecSignature 'PMrc'
#else
#define sndMidiRecSignature 'crMP'
\end{verbatim}

\textbf{SndSmfCallbacksType}

Structure that contains a set of application-defined functions that are called during MIDI playback. To register your callback functions, call \texttt{SndPlaySmf}.

\begin{verbatim}
typedef struct SndSmfCallbacksType {
  SndCallbackInfoType  completion;
  SndCallbackInfoType  blocking;
  SndCallbackInfoType  reserved;
} SndSmfCallbacksType;
\end{verbatim}

The fields are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>completion</td>
<td>Completion function; see \texttt{SndComplFuncType}.</td>
</tr>
<tr>
<td>blocking</td>
<td>Blocking function; see \texttt{SndBlockingFuncType}.</td>
</tr>
<tr>
<td>reserved</td>
<td>Reserved. Set this field to 0.</td>
</tr>
</tbody>
</table>
SndSmfChanRangeType
Structure that defines the range of enabled MIDI channels. Events on MIDI channels outside the enabled range are ignored. By default, no channels are enabled.

typedef struct SndSmfChanRangeType {
    UInt8  bFirstChan;
    UInt8  bLastChan;
} SndSmfChanRangeType;

The fields are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bFirstChan</td>
<td>The first enabled channel in the range [0, 15].</td>
</tr>
<tr>
<td>bLastChan</td>
<td>The last enabled channel in the range [0, 15].</td>
</tr>
</tbody>
</table>

WARNING! The SndSmfChanRangeType structure expects MIDI channels to be in the range [0, 15]; real MIDI channel values are in the range [1, 16]. Thus, PalmSource MIDI channel 0 is real MIDI channel 1, PalmSource MIDI channel 1 is real MIDI channel 2, and so on.

SndSmfCmdEnum
Constants used to tell SndPlaySmf whether to set or retrieve data.

typedef enum SndSmfCmdEnumTag {
    sndSmfCmdPlay = 1,
    sndSmfCmdDuration,
} SndSmfCmdEnum;

See SndPlaySmf for descriptions of these values.

SndSmfOptionsType
Structure that establishes MIDI performance parameters. This structure is used in the SndPlaySmf function to establish new parameter settings or to return the currently set values, depending
on how the function is called. In the case where the structure returns values, only the “performance marker” fields (dwStartMilliSec and dwEndMilliSec) are valid.

```c
typedef struct SndSmfOptionsType {
    UInt32  dwStartMilliSec;
    UInt32  dwEndMilliSec;
    UInt16  amplitude;
    Boolean interruptible;
    UInt8   reserved1;
    UInt32  reserved;
} SndSmfOptionsType;
```

The fields are

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwStartMilliSec</td>
<td>The “beginning of performance” marker, measured in milliseconds from the beginning of the track. A value of 0 plays the track from the beginning. The time difference between dwStartMilliSec and the performance time of the first subsequent MIDI event is respected. For example, if dwStartMilliSec is 2000 and the first (subsequent) note-on event is at 3000, there will be a 1000 millisecond “pause” before the note is played.</td>
</tr>
<tr>
<td>dwEndMilliSec</td>
<td>The “end of performance” marker, measured in milliseconds from the beginning of the track. To play to the end of the track, set this to_sndSmfPlayAllMilliSec.</td>
</tr>
</tbody>
</table>
**sndSmfPlayAllMilliSec**

Represents the (temporal) far end of a MIDI file. You can use this constant as the value of the `dwEndMilliSec` field of the `SndSmfOptionsType` structure before passing the structure to `SndPlaySmf`. This setting tells the function to play the entire file.

```c
#define sndSmfPlayAllMilliSec 0xFFFFFFFFUL
```

**SndSysBeepTag**

Constants that represent a set of pre-defined system beeps. See `SndPlaySystemSound` for more information on the system beeps and their intended uses.

```c
typedef enum SndSysBeepTag {
    sndInfo = 1,
    sndWarning,
    sndError,
    sndStartUp,
    sndAlarm,
    sndConfirmation,
    sndClick
} ;
```
Simple Sound Functions

SndCreateMidiList

**Purpose**
Generates a list of MIDI records.

**Declared In**
SoundMgr.h

**Prototype**

```c
Boolean SndCreateMidiList (UInt32 creator, Boolean multipleDBs, UInt16* recordCount, MemHandle *recordList)
```

**Parameters**

- **creator**
  Creator ID of the database in which the function looks for MIDI records. Pass 0 to search all databases.

- **multipleDBs**
  Pass true to search multiple databases for MIDI records. Pass false to search only in the first database that meets the search criteria.

- **recordCount**
  Returns the number of MIDI records that were found.

- **recordList**
  Returns a pointer to an array of SndMidiListItemType structures, one structure for each record that was found.

**Result**
Return true if records were found, otherwise returns false.

**Compatibility**
Implemented in Palm OS 3.0 and later.
**SndDoCmd**

**Purpose**
Asks the Sound Manager to perform a simple sound synthesis operation.

**Prototype**
```
Err SndDoCmd (void* channel, SndCommandPtr command, Boolean noWait)
```

**Parameters**
- `channel` A pointer to the sound channel on which you want to perform the operation. Pass NULL for the “shared” sound channel.
- `command` Pointer to a `SndCommandType` that describes the operation and associated parameters. See **Comments**, below, for information on the sound commands.
- `noWait` Sets the function to be asynchronous (true) or synchronous (false) with respect to the caller.

**IMPORTANT:** The Sound Manager only supports one channel of sound synthesis: You must pass NULL as the value of `channel`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>channel</code></td>
<td>A pointer to the sound channel on which you want to perform the operation. Pass NULL for the “shared” sound channel.</td>
</tr>
<tr>
<td><code>command</code></td>
<td>Pointer to a <code>SndCommandType</code> that describes the operation and associated parameters. See <strong>Comments</strong>, below, for information on the sound commands.</td>
</tr>
<tr>
<td><code>noWait</code></td>
<td>Sets the function to be asynchronous (true) or synchronous (false) with respect to the caller.</td>
</tr>
</tbody>
</table>

**IMPORTANT:** `SndDoCmd` is always synchronous: The `noWait` value is currently ignored.

**Result**
- `errNone` Success.
- `sndErrBadParam` Invalid parameter.
- `sndErrBadChannel` Invalid channel pointer.
- `sndErrQFull` The sound queue is full.
The sound operations that are performed by SndDoCmd are encapsulated in the `SndCommandType` structure:

```c
typedef struct SndCommandType {
    SndCmdIDType cmd;
    UInt8 reserved;
    Int32 param1;
    UInt16 param2;
    UInt16 param3;
} SndCommandType;
```

The `cmd` field represents the operation; the `paramN` fields are data that's passed to the operation. The operations and data that SndDoCmd supports are described below:

<table>
<thead>
<tr>
<th>cmd Value</th>
<th>param Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>sndCmdFreqDurationAmp</td>
<td><strong>param1</strong> is the tone's frequency in Hertz.</td>
</tr>
<tr>
<td></td>
<td><strong>param2</strong> is its duration in milliseconds.</td>
</tr>
<tr>
<td></td>
<td><strong>param3</strong> is its amplitude in the range [0, sndMaxAmp].</td>
</tr>
<tr>
<td></td>
<td>If the amplitude is 0, the sound isn't played and the function returns</td>
</tr>
<tr>
<td></td>
<td>immediately.</td>
</tr>
<tr>
<td>sndCmdFreqOn</td>
<td><strong>Same as the parameters for sndCmdFreqDurationAmp.</strong></td>
</tr>
<tr>
<td></td>
<td>The function returns immediately while the tone plays in the background.</td>
</tr>
<tr>
<td></td>
<td>Subsequent sound playback requests will interrupt the tone.</td>
</tr>
</tbody>
</table>

870  *Palm OS Programmer’s API Reference*
**Sound Manager**

**Simple Sound Functions**

### Compatibility

Commands other than `sndCmdFreqDurationAmp` are implemented in Palm OS 3.0 and later. In OS versions earlier than 3.0, `SndDoCmd` will crash with a fatal error if you pass a command other than `sndCmdFreqDurationAmp`.

### See Also

- **SndPlaySmf**

---

### SndGetDefaultVolume

**Purpose**

Returns volume levels cached by the Sound Manager.

**Prototype**

```c
void SndGetDefaultVolume (UInt16 *alarmAmp, UInt16 *sysAmp, UInt16 *masterAmp)
```

**Parameters**

- `- alarmAmp` Pointer to the alarm amplitude.
- `- sysAmp` Pointer to the system sound amplitude.
- `- masterAmp` Pointer to the master amplitude.

**Comments**

Pass NULL for settings that you don’t care about.

**Compatibility**

Never call this function in Palm OS 5. To retrieve default volume levels, you should ask for the user’s preferences settings.

**See Also**

- **SndSetDefaultVolume**
**SndInterruptSmfIrregardless**

**Purpose**
Stops the currently playing MIDI file, even if the performance was declared to be intolerant of interruptions.

**Prototype**
Err SndInterruptSmfIrregardless (void)

**Result**
Always returns errNone (success).

**Compatibility**
Implemented in Palm OS 4.0 and later. The name is incompatible with the English language.

**See Also**
SndPlaySmf, SndPlaySmfResourceIrregardless

**SndPlaySmf**

**Purpose**
Performs a Standard MIDI File, or returns the duration of the file.

**Prototype**
Err SndPlaySmf (void *channel, SndSmfCmdEnum command, UInt8 *midiData, SndSmfOptionsType *options, SndSmfChanRangeType *channelRange, SndSmfCallbacksType *callbacks, Boolean noWait)

**Parameters**
- **-> channel**
  A pointer to the sound channel on which you want to perform the MIDI file. Pass NULL for the “shared” sound channel.

**IMPORTANT:** The Sound Manager only supports one channel of sound synthesis: You must pass NULL as the value of channel.

- **-> command**
  Either SndSmfCmdPlay (play the file) or SndSmfCmdDuration (return the duration of the file in milliseconds).

- **-> midiData**
  The MIDI data; this can point to an SndMidiRecHdrType struct, or it can point
Sound Manager
Simple Sound Functions

directly to the actual MIDI data bytes in memory.

-> options A pointer to a SndSmfOptionsType that defines performance parameters (volume, starting offset, interruption tolerance). For default behavior, pass NULL. For more information (including default settings), see SndSmfOptionsType.

-> channelRange A pointer to a SndSmfChanRangeType that specifies the range of MIDI channels (in the SMF data) to use during playback. To play all channels, pass NULL.

-> callbacks A pointer to a SndSmfCallbacksType that holds your callback functions. Pass NULL if you don’t want any callbacks.

noWait This value is ignored. This function always finishes playing the SMF selection before returning (but see Comments, below).

Result errNone Success.
sndErrBadParam Invalid value passed to this function.
sndErrBadChannel Invalid sound channel.
sndErrMemory Insufficient memory.
sndErrOpen Tried to open channel that’s already open.
sndErrQFull Can’t accept more notes.
sndErrFormat Unsupported data format.
sndErrBadStream Invalid data stream.
sndErrInterrupted Play was interrupted.
Comments

Although this call is always synchronous, you can register a “blocking” function that’s called periodically as the MIDI file is playing. See `SndBlockingFuncType` for more information.

Normally, playback is halted by events generated by user interaction with the screen, digitizer, or hardware-based buttons. You can override this behavior by setting the `interruptible` field of the `options` argument to false.

This function waits until any currently playing simple sound has finished before starting playback of the requested MIDI data. A similar function, `SndPlaySmfIrregardless`, doesn’t wait: It interrupts the current performance and immediately begins playback of the requested data.

Compatibility

Implemented in Palm OS 3.0 and later.

**SndPlaySmfIrregardless**

Purpose

Like `SndPlaySmf`, but interrupts any currently playing simple sound, regardless of that sound’s declared interruption tolerance.

Prototype

```c
Err SndPlaySmfIrregardless (void *channel, SndSmfCmdEnum command, UInt8 *midiData, SndSmfOptionsType *options, SndSmfChanRangeType *channelRange, SndSmfCallbacksType *callbacks, Boolean noWait)
```

Comments

For further information, see `SndPlaySmf`.

Compatibility

Implemented in Palm OS 4.0 and later.
SndPlaySmfResource

**Purpose**
Plays a MIDI track read out of an open resource database.

**Prototype**
Err SndPlaySmfResource (UInt32 resType, Int16 resID, SystemPreferencesChoice volume)

**Parameters**
- resType: SMF resource type.
- resID: SMF resource ID.
- volume: Volume setting; one of:
  - prefSysSoundVolume
  - prefGameSoundVolume
  - prefAlarmSoundVolume

**Result**
- errNone: Success.
- sndErrBadParam: The volumeSelector parameter has an invalid value or the SMF resource has invalid data.
- dmErrCantFind: The specified resource doesn’t exist.
  
**Comments**
This function plays the entire MIDI file using all MIDI channels. Playback is interrupted by a key down or digitizer event. No callbacks are specified.

This function waits until any currently playing simple sound has finished before starting playback of the requested MIDI data. A similar function, SndPlaySmfResourceIgnored, doesn’t wait: It interrupts the current performance and immediately begins playback of the requested data.

**Compatibility**
Implemented in Palm OS 3.2 and later.
SndPlaySmfResourceIrregardless

**Purpose**
Like SndPlaySmfResource, but interrupts any currently playing simple sound, regardless of that sound’s declared tolerance for interruption.

**Prototype**
Err SndPlaySmfResourceIrregardless (UInt32 resType, Int16 resID, SystemPreferencesChoice volumeSelector)

**Comments**
For further information, see SndPlaySmfResource.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

SndPlaySystemSound

**Purpose**
Plays a pre-defined (simple) system sound.

**Prototype**
void SndPlaySystemSound (SndSysBeepType beepID)

**Parameters**
-> beepID One of the system beep sound constants, listed below.

**Comments**
The system beep sounds are represented by the SndSysBeepType constants:

- sndInfo. Heralds non-crucial information.
- sndWarning. Grabs the user’s attention
- sndError. Indicates an illegal operation.
- sndStartUp. Played at device start up time.
- sndAlarm. Generic alarm sound; note that this is not the Datebook’s alarm sound.
- sndConfirmation. Indicates approval or acceptance.
- sndClick. The button click sound.

If you’re playing an alarm (sndAlarm), the user’s alarm volume preference setting is used. For all other system sounds, the system volume preference is used.
In addition, sndAlarm sounds are played synchronously (the function blocks). All other sounds are played asynchronously.

### SndSetDefaultVolume

**Purpose**
Sets the default sound volume levels cached by the Sound Manager.

**Prototype**
```c
void SndSetDefaultVolume (UInt16 *alarmAmp,
                          UInt16 *sysAmp,
                          UInt16 *masterAmp)
```

**Parameters**
- **alarmAmp**
  Pointer to the alarm amplitude.
- **sysAmp**
  Pointer to the system sound amplitude.
- **masterAmp**
  Pointer to the master amplitude.

**Result** Returns nothing.

**Comments**
Any of the arguments may be NULL. In that case, the corresponding setting is not altered.

**Compatibility**
Never call this function in Palm OS 5.

**See Also**
[SndGetDefaultVolume](#)

---

### Simple Sound Application-Defined Functions

#### SndBlockingFuncType

**Purpose**
Invoked periodically during SMF playback.

**Prototype**
```c
Boolean SndBlockingFuncType (void* channel,
                            UInt32 userData, Int32 time)
```

**Parameters**
- **channel**
  A pointer to the sound channel on which the file is being played. Currently always NULL.
- **userData**
  Arbitrary data that’s defined when the callback function is registered.
Sound Manager
Simple Sound Application-Defined Functions

- time
  The amount of time, in milliseconds, available for completion of this function.

Result
If the function returns true, playback continues. If it returns false, playback is aborted.

Comments
Your SndBlockingFuncType function is called whenever the MIDI parser is “between notes.” You can do whatever you want during this period, as long as it doesn’t take more than time milliseconds.

See Also
SndComplFuncType for an example of how to register MIDI callback functions.

SndComplFuncType

Purpose
Invoked immediately after a MIDI file (SMF) finishes playing.

Prototype
void SndComplFuncType (void* channel,
UInt32 userData)

Parameters
- channel
  A pointer to the sound channel on which the file was played; currently always NULL.

- userData
  Caller-defined data that’s copied when the callback is registered.

Comments
To register a SndComplFuncType function (and the userData that’s fed to it), you define a SndCallbackInfoType structure, load that structure into the completion field of a SndSmfCallbacksType structure, and then pass that structure to SndPlaySmf.

The following example shows how to register a completion function as well as an SMF “blocking” function (SndBlockingFuncType).

/* First, we create and load two SndCallbackInfoType structs. We assume that MyCompletionFunc is a valid SndComplFuncType implementation, and that MyBlockingFunc is a valid SndBlockingFuncType.
 */
SndCallbackInfoType completionFunc, blockingFunc;
completionFunc.funcP = MyCompletionFunc;
completionFunc.dwUserData = 47;
blockingFunc.funcP = MyBlockingFunc;
blockingFunc.dwUserData = (UInt32)&(char *)"we're done";

/* Next, we create an SndSmfCallbacksType struct and load the previous structs into it. */
SndSmfCallbacksType callbacks;
callbacks.completion = completionFunc
callbacks.blocking = blockingFunc;

/* Finally, we pass the callbacks into SndPlaySmf. */
SndPlaySmf(NULL, sndSmfCmdPlay, smf, NULL, NULL, callbacks, true);

### Sampled Sound Structures, Constants, and Data Types

**New**

**SndPtr**

Type used to cast a pointer to the sound data used by `SndPlayResource`.

```
typedef void *SndPtr;
```

**Compatibility**

Implemented if [Sound Stream Feature Set](#) is present.

**New**

**SndSampleType**

Data type that’s used for `SndSampleTypeTag` values.

```
typedef Enum16 SndSampleType;
```

**Compatibility**

Implemented if [Sound Stream Feature Set](#) is present.
**New**

**SndSampleTypeTag**

Constants that represent the sample format (size, data type, endianness) of a sampled sound stream. Used by `SndStreamCreate`.

The enumeration is defined as:

```c
typedef enum SndSampleTypeTag {
    sndInt8 = 0x01,
    sndUInt8 = 0x11,
    sndInt16Big = 0x02,
    sndInt16Little = 0x12,
    sndInt32Big = 0x04,
    sndInt32Little = 0x14,
    sndFloatBig = 0x24,
    sndFloatLittle = 0x34,
    #if CPU_ENDIAN == CPU_ENDIAN_BIG
        sndInt16 = sndInt16Big,
        sndInt16Opposite = sndInt16Little,
        sndInt32 = sndInt32Big,
        sndInt32Opposite = sndInt32Little,
        sndFloat = sndFloatBig,
        sndFloatOpposite = sndFloatLittle
    #else
        sndInt16 = sndInt16Little,
        sndInt16Opposite = sndInt16Big,
        sndInt32 = sndInt32Little,
        sndInt32Opposite = sndInt32Big,
        sndFloat = sndFloatLittle,
        sndFloatOpposite = sndFloatBig
    #endif
};
```

The constants are:

<table>
<thead>
<tr>
<th>_sndInt8, sndUInt8</th>
<th>Signed and unsigned 8-bit data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>sndInt16, sndInt32, sndFloat</td>
<td>16-bit integer, 32-bit integer, and floating point data in the device’s native endianness.</td>
</tr>
</tbody>
</table>
Note that the lower four bits of these constants gives the size (in bytes) of a single sample, thus:

\[ \text{UInt8 byteSize} = \text{formatConstant} \& \ 0x0f \]

**NOTE:** In Palm OS 5, the 32-bit and floating point formats aren’t supported.

### Compatibility
Implemented if Sound Stream Feature Set is present.

#### New
**SndStreamMode**
Data type that’s used for SndStreamModeTag values.

```c
typedef Enum8 SndStreamMode;
```

### Compatibility
Implemented if Sound Stream Feature Set is present.

#### New
**SndStreamModeTag**
Constants that represent the “direction” (input or output) of a sampled sound stream. Used by the SndStreamCreate function.

```c
typedef enum SndStreamModeTag {
```
The constants are:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sndInput</td>
<td>Input stream used for recording.</td>
</tr>
<tr>
<td>sndOutput</td>
<td>Output stream used for playback.</td>
</tr>
</tbody>
</table>

**Compatibility**

Implemented if [Sound Stream Feature Set](#) is present.

---

**New**

**SndStreamRef**

Data type that represents a sampled stream. You create an SndStreamRef through `SndStreamCreate`.

```c
typedef UInt32 SndStreamRef;
```

**Compatibility**

Implemented if [Sound Stream Feature Set](#) is present.

---

**New**

**SndStreamWidth**

Data type that’s used for `SndStreamWidthTag` values.

```c
typedef Enum8 SndStreamWidth;
```

**Compatibility**

Implemented if [Sound Stream Feature Set](#) is present.

---

**New**

**SndStreamWidthTag**

Constants that represent mono and stereo sampled data streams. Used by the `SndStreamCreate` function.
typedef enum SndStreamWidthTag {
    sndMono,
    sndStereo
};

The constants are:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sndMono</td>
<td>Mono (one channel) stream.</td>
</tr>
<tr>
<td>sndStereo</td>
<td>Stereo (two channel) stream.</td>
</tr>
</tbody>
</table>

**Compatibility** Implemented if Sound Stream Feature Set is present.

### New

**Sound Resource Playback Flags**

Flags used by `SndPlayResource`.

```c
#define sndFlagSync 0x00000000
#define sndFlagAsync 0x00000001
#define sndFlagNormal sndFlagSync
```

See `SndPlayResource` for information on these flags.

**Compatibility** Implemented if Sound Stream Feature Set is present.

### New

**Stereo Pan Constants**

The stereo pan settings can be used in `SndStreamSetPan`.

```c
#define sndPanCenter (0)
#define sndPanFullLeft (-1024)
#define sndPanFullRight (1024)
```

**Compatibility** Implemented if Sound Stream Feature Set is present.
New Volume Constants

The volume constants can be used by `SndStreamSetVolume` and `SndPlayResource`. The constants tell the functions to retrieve the named sound volume preference (as set by the user) and apply it as a volume setting.

```c
enum {
    sndSystemVolume = -1,
    sndGameVolume = -2,
    sndAlarmVolume = -3
};
```

Compatibility

Implemented if Sound Stream Feature Set is present.

Sampled Sound Functions

New `SndPlayResource`

**Purpose**

Plays formatted sound data read from a resource or file.

**Prototype**

```c
Err SndPlayResource (SndPtr sound, Int32 ampScale, UInt32 flags)
```

**Parameters**

- `-> sound` A pointer to the beginning of the formatted sound (header and all). Currently, only WAVE data is recognized (see Comments, below); in this case, sound must point to the "RIFF" ID (byte 0 in a simple .wav file).

- `-> ampScale` Amplitude scalar, in the range [0, 32k]. See `SndStreamSetVolume` for information on how amplitude scalar values are applied.

- `-> flags` Settings flags. Currently, the only setting is function synchronization: Choose between
sndFlagSync and sndFlagAsync. The former tells the function to wait until all sound data has been fed to the DAC before returning (i.e. the function will return just a bit before the sound has finished playing). The sndFlagAsync flag tells the function to return immediately while playback continues in a separate thread.

As a convenience, the sndFlagNormal value is a shorthand for the set of “normal” flag settings. Currently, this is set to sndFlagSync.

**Result**

- **errNone**: Success.
- **sndErrBadParam**: `sound` contains no data.
- **sndErrFormat**: The data is in an unsupported format.
- **sndErrMemory**: The function couldn’t allocate sufficient memory.
- **other errors**: The device couldn’t allocate system resources for the sound.

**Comments**

Supported WAVE parameters are:

- uncompressed (PCM) or IMA 4-bit adaptive differential (IMA ADPCM). The ADPCM type is also known as DVI ADPCM; in a WAVE file, it’s known as format 0x11.

  - One or two-channels
  - All normal sampling rates (8k, 11k, 22.05k, 44.1k, 48, 96k).

You can’t interrupt or abort a resource playback once it’s been initiated. The resource always play to the end of the data.

**Compatibility**

Implemented if Sound Stream Feature Set is present.
**New**

**SndStreamCreate**

**Purpose**

Creates a new audio data stream that can be used to record or playback uncompressed, sampled audio data.

**Prototype**

```c
Err SndStreamCreate (SndStreamRef *stream,
                     SndStreamMode mode, UInt32 sampleRate,
                     SndSampleType type, SndStreamWidth width,
                     SndStreamBufferCallback callback,
                     void *callbackArg, UInt32 bufferSize,
                     Boolean callbackIsARM)
```

**Parameters**

- `<- stream` Token that represents the newly created stream.
- `-> mode` Constant that represents the “direction” of the data. Either `sndInput` (for recording), or `sndOutput` (for playback).
- `-> sampleRate` Sampling rate, in frames-per-second. The value passed here is the native rate of the data, given as a number (22050, 44100, 48000, etc.). The maximum rate is 96000.
- `-> type` Sample quantization and endianness (but see the section on “Data Formats,” below).
- `-> width` A constant that represents the number of channels of data in the stream; either `sndMono` or `sndStereo`.
- `-> callback` A callback function that gets called when another buffer of data is needed.
- `-> callbackArg` Caller-defined data that gets passed to `callback`.
- `-> bufferSize` Preferred size (in frames) for the buffers that are passed to `callback`. Note that the actual size may be different.
-> callbackIsARM
(68k only) Pass true if the callback function is written in ARM-native code; if it’s 68k, pass false.

Result

errNone  Success.
sndErrBadParam  stream is invalid, bufferFunc is NULL, the sampleRate is too high (greater than 96000), or the device doesn’t like some other sound parameter value.
sndErrorMemory  All streams are being used (there is a maximum of 16), or memory for this stream couldn’t otherwise be allocated.
other errors  The device couldn’t allocate system resources for the stream.

Comments

This function creates a new audio stream into which you can write (playback) or from which you can read (record) buffers of uncompressed, sampled audio data. The stream’s “direction”—whether it will be used for recording or playback—is described by the mode argument.

You can create one input stream and as many as 15 output streams. The “active” end of a stream is hardwired to read from or write to the device’s sound driver. This means you can’t “redirect” an input stream to read from a file (for example), nor can you connect one output stream to another output stream in an attempt to create a filter chain. You can, however, collect data from the input stream, manipulate it, and then write it to an output stream.

Data Formats

The format of the data that flows through the stream is described by the sampleRate, type, and width arguments:

- The data format that you specify for an input stream must match the data that’s produced by the audio hardware.
For an output stream, you can specify any of the formats that the Sound Manager supports; the data is automatically converted to the output hardware’s native audio format. Whether your stream’s format setting actually affects the hardware is undefined. For example, if you set an output stream to use a 48k sampling rate, that doesn’t mean that the DAC will be set to 48k.

If you look at the `SndSampleTypeTag` constants, you’ll see three flavors for each quantization type: There’s a big-endian version, a little-endian version, and a native-endian version (defined as one of the other two). In general, you should use the native-endian version when choosing a value for the `type` parameter. The one exception to this (in Palm OS 5) is if your application is written for 68k, but uses an ARM callback function (i.e. `callbackIsARM == true`). In this case you should use one of the little-endian formats as the `type` value when you create your stream.

**Running the Stream**

The new stream starts running when you pass the `stream` token returned by this function to the `SndStreamStart` function. This initiates a series of calls to your callback function, which is where the action is: Each callback invocation is passed a buffer into which you write or from which you read a chunk of audio data. The callback function is also passed the `callbackArg` that you supply here. See `SndStreamBufferCallback` for more information on the callback function.

**Buffering and Latency**

Currently, audio streams are double-buffered. With regard to playback, this means that while one buffer (buffer A) is being played, your callback function is placing data in the other buffer (B). When A is “empty,” the Sound Manager seamlessly starts playing buffer B, and passes buffer A back to your callback; when B is empty, the Manager start playing A, and passes back B, and so on. It’s important that your callback function fills the data buffers as quickly as possible—certainly no longer than it takes to play a buffer of data. This same double-buffer scheme is also applied to sound recording although, of course, for recording you’re emptying each buffer (and doing something with the data) in your callback function.
Regarding latency, you can use the `bufferSize` argument to suggest a buffer size and thereby increase or decrease latency, but you can't change the number of buffers. Keep in mind that the actual buffer size that's used may not be the same as the size you suggest; hardware and memory limitations may enforce a maximum or minimum buffer size. Also keep in mind that the `bufferSize` is measures in frames (not bytes).

*SndStreamStart, SndStreamDelete, SndStreamBufferCallback*

**Compatibility**

Implemented if [Sound Stream Feature Set](#) is present.

---

**New**

**SndStreamDelete**

**Purpose**

Stops the stream and destroys it.

**Prototype**

```c
Err SndStreamDelete (SndStreamRef stream)
```

**Parameters**

`-> stream` Stream token, as returned through [SndStreamCreate](#).

**Result**

`errNone` Success.

`sndErrBadParam` `stream` is invalid.

**Comments**

[SndStreamStop](#) is called before the stream is destroyed. You should never call this function as part of the implementation of a callback function.

**Compatibility**

Implemented if [Sound Stream Feature Set](#) is present.
New SndStreamGetPan

Purpose
Retrieves a stream’s stereo balance.

Prototype
Err SndStreamGetPan (SndStreamRef stream, Int32 *pan)

Parameters
-> stream Stream token, as returned through SndCreateRawStream.
<- pan Pan value in the range [-1024 (hard left), 1024 (hard right)]. Center balance is 0.

Result
errNone Success.
sndErrBadParam Invalid stream, or pan is NULL.

Compatibility
Implemented if Sound Stream Feature Set is present.

See Also
SndStreamSetPan

New SndStreamGetVolume

Purpose
Retrieves the amplitude scalar for a sound stream.

Prototype
Err SndStreamGetVolume (SndStreamRef stream, Int32 *ampScale)

Parameters
-> stream Stream token, as returned through SndCreateRawStream.
<- ampScale Amplitude scalar, in the range [0, 32k]. See SndStreamSetVolume for more information.

Result
errNone Success.
**SndErrBadParam**

Invalid stream, or volume is NULL.

**See Also**  
[SndStreamSetVolume](#)

**Compatibility**  
Implemented if [Sound Stream Feature Set](#) is present.

---

**New**  

**SndStreamPause**

**Purpose**  
Pauses and resumes a sample stream.

**Prototype**  
Err SndStreamPause (SndStreamRef stream, Boolean pause)

**Parameters**

- `-> stream`  
  Stream token, as returned by [SndStreamCreate](#).

- `-> pause`  
  If true, the function pauses the stream; if it’s false, it resumes the stream.

**Result**

- `errNone`  
  Success. Note that `errNone` is returned even if the stream is already in the requested state.

- `sndErrBadParam`  
  Invalid stream.

**Comments**

Currently, SndStreamPause is implemented through calls to [SndStreamStop](#) and [SndStreamStop](#) (the former if `pause`==true; the latter if `pause`==false). See those functions for details about “pausing” and “resuming” a sound stream. You can’t nest pauses; a single resume request is effective, regardless of the number of times the stream has been told to pause.

**Compatibility**

Implemented if [Sound Stream Feature Set](#) is present.
New

SndStreamSetPan

Purpose  Sets a stream’s stereo balance.

Prototype  

Err SndStreamSetPan (SndStreamRef stream,
Int32 pan);

Parameters  

-> stream  Stream token, as returned through
SndCreateRawStream.

-> pan  Pan value in the range [-1024 (full left), 1024
(full right)]. Center balance is 0. As a
convenience, you can use the values described
in “Stereo Pan Constants.” Note that values
outside of the valid range may yield
unexpected results (but don’t generate an
error).

Result  

errNone  Success.

sndErrBadParam  Invalid stream.

Comments  The pan value is used as a scalar on a channel’s volume such that a channel
increases from 0 (inaudible) to full volume as the pan value moves from an
extreme to 0. Graphically, it looks like this:
**Compatibility**
Implemented if Sound Stream Feature Set is present.

**See Also**
SndStreamGetPan

### New

**SndStreamSetVolume**

**Purpose**
Sets the amplitude scalar for a sound stream.

**Prototype**
```c
Err SndStreamSetVolume (SndStreamRef stream,
Int32 ampScale);
```

**Parameters**
- `stream` Stream token, as returned through SndCreateRawStream.
- `ampScale` Amplitude scalar in the range [0, 32k]. Values less than 0 are converted to 1024 (unity gain).

**Result**
- `errNone` Success.
- `sndErrBadParam` Invalid stream.

**Comments**
The `ampScale` value is applied as an amplitude scalar on the samples that this stream’s callback function produces. The scalar is...
in the range [0, 32k], where 1024 is unity gain (i.e. the samples are multiplied by 1.0). The mapping of ampScale to scalar is linear; thus a volume of 512 scales the samples by ~.5, 2048 scales by ~2.0, and so on.

To specify a user preference volume setting, use one of sndSystemVolume, sndGameVolume, or sndAlarmVolume. These values are guaranteed to be less than unity gain.

If the stream is stereo, both channels are scaled by the same amplitude scalar. To adjust the balance between the channels, use SndStreamSetPan.

### Compatibility
Implemented if Sound Stream Feature Set is present.

### New SndStreamStart

**Purpose**
Starts a sample stream running.

**Prototype**
Err SndStreamStart (SndStreamRef stream);

**Parameters**
-> stream Stream token, as returned by SndStreamCreate.

**Result**
errNone Success. Note that errNone is returned even if the stream is already running.

sndErrBadParam stream is invalid.

**Comments**
If the stream is already running, the function returns immediately (with errNone). If it isn’t running, the function starts the stream by initiating invocations of its callback function. If it’s paused (through SndStreamPause), the stream is resumed.

You can call this function from some other stream’s callback function. In other words, a stream can tell another stream to start playing.
New SndStreamStop

Purpose
Stops a sample stream running.

Prototype
Err SndStreamStop (SndStreamRef stream);

Parameters
-> stream Stream token, as returned by SndStreamCreate.

Result
errNone Success. Note that errNone is returned even if the stream has already been stopped.
sndErrBadParam stream is invalid.

Comments
Stops a running sound stream by neglecting to call its callback function. The stream remains in this suspended state until you call SndStreamStart.

You can call this function from the stream’s own callback function. In other words, a stream can stop itself.

Compatibility
Implemented if Sound Stream Feature Set is present.
Sampled Sound Application-Defined Functions

New

SndStreamBufferCallback

Purpose
Used to deliver a data buffer from/into which you read/write sound data.

Prototype
Err SndStreamBufferCallback (void *userData, SndStreamRef stream, void *buffer, UInt32 frameCount);

Parameters
- **userData**: Caller-defined data, as provided by the callbackArg parameter to SndStreamCreate.
- **stream**: Token that represents the stream that this buffer belongs to.
- **buffer**: The data buffer.
- **frameCount**: Number of sample frames the buffer contains.

Result
Currently, the return value is ignored.

Comments
If this is an input (recording) stream, you read the data in buffer. If this is an output (playback) stream, you write data into buffer. In either case, you want to do this as quickly as possible to avoid data underflow.

Note that the function doesn’t tell you anything about the format of the data. You can use the userData argument to pass this information into the function.

The callback function is executed in a task that’s created and managed by the Sound Manager. Because of this, the function doesn’t have access to the symbols that you create in your application. Again, use the userData argument if you need to pass pointers to your symbols.
**Compatibility**
Implemented if Sound Stream Feature Set is present.

**See Also**
SndStreamCreate
Sound Manager
Sampled Sound Application-Defined Functions
This chapter provides reference material for the standard IO API:

- **Standard IO Functions**
- **Standard IO Provider Functions**
- **Application-Defined Function**

The header files `StdIOPalm.h` and `StdIOProvider.h` declare the standard IO API. For more information on using the standard IO API, see the chapter “**Standard IO Applications**” in the *Palm OS Programmer’s Companion*, vol. I.

**Standard IO Functions**

The macros and functions in this section enable standard IO.

**fgetc**

**Purpose**
Macro that calls `Siofgetc` to return the next character from the input stream.

**Declared In**
`StdIOPalm.h`

**Prototype**
`fgetc (fs)`

**Parameters**
`-> fs` An input stream from which to read the next character. You can specify only the value `stdin` for this parameter; alternate streams are not currently implemented.

**Result**
The next character from the input stream. The return value `EOF` indicates an error occurred.
**fgets**

**Purpose**
Macro that calls *Siofgets* to return a string from the input stream.

**Declared In**
*StdIOPalm.h*

**Prototype**
`fgets (strP, maxChars, fs)`

**Parameters**
- `<strP>`
  A pointer to the returned string.
- `-> maxChars`
  The number of characters to read from the input stream, plus one for the null terminator.
- `-> fs`
  An input stream from which to read the next character. You can specify only the value *stdin* for this parameter; alternate streams are not currently implemented.

**Result**
A pointer to the string read from the input stream. If an error or EOF occurs before any characters are read, returns `NULL`.

**Comments**
The returned string is always terminated by a null character.

**fprintf**

**Purpose**
Macro that calls *Siofprintf* to write formatted output to an output stream.

**Declared In**
*StdIOPalm.h*

**Prototype**
`fprintf (fs, formatP, ...)`

**Parameters**
- `-> fs`
  An output stream to which to write the formatted output. You can specify only the value *stdout* for this parameter; alternate streams are not currently implemented.
- `-> formatP`
  A pointer to a format string that controls how subsequent arguments are converted for output.
Zero or more parameters to be formatted as specified by the formatP string.

**Result**

Returns the number of characters written out (not including the null terminator used to end output strings). Returns a negative number if there is an error.

**Comments**

This function internally calls StrVPrintP to do the formatting. See that function for details on which format specifications are supported.

### fputs

**Purpose**

Macro that calls Siofputc to write a character to the output stream.

**Declared In**

StdIOPalm.h

**Prototype**

fputc (c, fs)

**Parameters**

- `-> c` A character to write to the output stream.
- `-> fs` An output stream to which to write the character. You can specify only the value stdout for this parameter; alternate streams are not currently implemented.

**Result**

The character that was written. If an error occurs, the value EOF is returned.
**fputs**

**Purpose**
Macro that calls `Siof.puts` to write a string to the output stream.

**Declared In**
`StdIOPalm.h`

**Prototype**
`fputs (strP, fs)`

**Parameters**
- `-> strP` A pointer to the string to write.
- `-> fs` An output stream to which to write the string. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.

**Result**
Returns 0 on success and the value `EOF` on error.

**getchar**

**Purpose**
Macro that calls `Siof.getc` to read the next character from the `stdin` input stream.

**Declared In**
`StdIOPalm.h`

**Prototype**
`getchar ()`

**Result**
The next character from the input stream. The return value `EOF` indicates an error occurred.
gets

**Purpose**
Macro that calls `siogets` to read a string from the `stdin` input stream.

**Declared In**
`StdIOPalm.h`

**Prototype**
`gets (strP)`

**Parameters**
- `<strP>` A pointer to the returned string.

**Result**
A pointer to the string read from the input stream. If an error or EOF occurs before any characters are read, returns `NULL`.

**Comments**
The returned string does not include a null terminator. You must ensure that the input line, if any, is sufficiently short to fit in the string.

printf

**Purpose**
Macro that calls `sioprintf` to write formatted output to the `stdout` output stream.

**Declared In**
`StdIOPalm.h`

**Prototype**
`printf (formatP, ...)`

**Parameters**
- `formatP` A pointer to a format string that controls how subsequent arguments are converted for output.
- `...` Zero or more parameters to be formatted as specified by the `formatP` string.

**Result**
Returns the number of characters written out (not including the null terminator used to end output strings).
Comments
This function internally calls _StrVPRINTF_ to do the formatting. See that function for details on which format specifications are supported. Returns a negative number if there is an error.

**putc**

**Purpose**
Macro that calls _Siofputc_ to write a character to the output stream.

**Declared In**
StdIOPalm.h

**Prototype**
putc (c, fs)

**Parameters**
- `-> c` A character to write to the output stream.
- `-> fs` An output stream to which to write the character. You can specify only the value stdout for this parameter; alternate streams are not currently implemented.

**Result**
The character that was written. If an error occurs, the value EOF is returned.

**putchar**

**Purpose**
Macro that calls _Siofputc_ to write a character to the stdout output stream.

**Declared In**
StdIOPalm.h

**Prototype**
putchar (c)

**Parameters**
- `-> c` A character to write to the stdout output stream.

**Result**
The character that was written. If an error occurs, the value EOF is returned.
puts

**Purpose**  
Macro that calls *Sioputs* to write a string to the output stream stdout.

**Declared In**  
*StdIOPalm.h*

**Prototype**  
`puts (strP)`

**Parameters**  
- `strP`  
  A pointer to the string to write to stdout.

**Result**  
Returns a nonnegative value on success and the value `EOF` on error.

---

SioAddCommand

**Purpose**  
Adds a built-in command that is supplied by the standard IO provider application.

**Declared In**  
*StdIOPalm.h*

**Prototype**  
`void SioAddCommand (const Char *cmdStr, SioMainProcPtr cmdProcP)`

**Parameters**  
- `cmdStr`  
  Pointer to a string that is the command name.
- `cmdProcP`  
  Pointer to the command entry point function (the *SioMain* function).

**Result**  
Returns nothing.

**Comments**  
This routine is useful for registering a command that is inside the standard IO provider application instead of in its own database. This routine must be used to test commands under the Simulator since it can’t launch application databases.
**Siofgetc**

Return the next character from the input stream.

**Declared In**  StdIOPalm.h

**Prototype**  Int16 Siofgetc (FILE *fs)

**Parameters**

- `-> fs`  An input stream from which to read the next character. You can specify only the value `stdin` for this parameter; alternate streams are not currently implemented.

**Result**  The next character from the input stream. The return value `EOF` indicates an error occurred.

**See Also**  `fgetc`

---

**Siofgets**

**Purpose**  Return a string from the input stream.

**Declared In**  StdIOPalm.h

**Prototype**  Char *Siofgets (Char *strP, UInt16 maxChars, FILE *fs)

**Parameters**

- `<- strP`  A pointer to the returned string.
- `-> maxChars`  The number of characters to read from the input stream, plus one for the null terminator.
- `-> fs`  An input stream from which to read the next character. You can specify only the value `stdin` for this parameter; alternate streams are not currently implemented.

**Result**  A pointer to the string read from the input stream. If an error or EOF occurs before any characters are read, returns `NULL`. 
Siofprintf

Purpose
Write formatted output to an output stream.

Declared In
StdIOPalm.h

Prototype
Int16 Siofprintf (FILE *fs, const Char *formatP, ...)

Parameters
- > fs
An output stream to which to write the formatted output. You can specify only the value stdout for this parameter; alternate streams are not currently implemented.

- > formatP
A pointer to a format string that controls how subsequent arguments are converted for output.

- > ...
Zero or more parameters to be formatted as specified by the formatP string.

Result
Returns the number of characters written out (not including the null terminator used to end output strings). Returns a negative number if there is an error.

Comments
This function internally calls StrVPrintF to do the formatting. See that function for details on which format specifications are supported.

See Also
fgets
fprintf
Siofputc

**Purpose**
Write a character to the output stream.

**Declared In**
StdIOPalm.h

**Prototype**
`Int16 Siofputc (Int16 c, FILE *fs)`

**Parameters**
- `-> c`
  A character to write to the output stream.
- `-> fs`
  An output stream to which to write the character. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.

**Result**
The character that was written. If an error occurs, the value `EOF` is returned.

**See Also**
`fputc`

Siofputs

**Purpose**
Write a string to the output stream.

**Declared In**
StdIOPalm.h

**Prototype**
`Int16 Siofputs (const Char *strP, FILE *fs)`

**Parameters**
- `-> strP`
  A pointer to the string to write.
- `-> fs`
  An output stream to which to write the string. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.

**Result**
Returns 0 on success and the value `EOF` on error.

**See Also**
`fputs`
Siogets

Purpose    Read a string from the stdin input stream.

Declared In     StdIOPalm.h

Prototype     Char *Siogets (Char *strP)

Parameters     <- strP          A pointer to the returned string.

Result         A pointer to the string read from the input stream. If an error or EOF
                occurs before any characters are read, returns NULL.

Comments       The returned string does not include a null terminator. You must
                ensure that the input line, if any, is sufficiently short to fit in the
                string.

See Also       gets

Siopprintf

Purpose       Write formatted output to the stdout output stream.

Declared In   StdIOPalm.h

Prototype     Int16 Siopprintf (const Char *formatP, ...)

Parameters     -> formatP    A pointer to a format string that controls how
                subsequent arguments are converted for output.
                -> ...                  Zero or more parameters to be formatted as
                specified by the formatP string.

Result         Returns the number of characters written out (not including the null
                terminator used to end output strings).
Comments

This function internally calls `StrVPrintf` to do the formatting. See that function for details on which format specifications are supported. Returns a negative number if there is an error.

See Also

`printf`

Sioputs

Purpose

Write a string to the output stream `stdout`.

Declared In

`StdIOPalm.h`

Prototype

`Int16 Sioputs (const Char *strP)`

Parameters

`-> strP` A pointer to the string to write to `stdout`.

Result

Returns a nonnegative value on success and the value `EOF` on error.

See Also

`puts`

Siosystem

Purpose

Execute another Stdio command.

Declared In

`StdIOPalm.h`

Prototype

`Int16 Siosystem (const Char *cmdStrP)`

Parameters

`-> cmdStrP` A pointer to a string containing the command line to execute.

Result

Returns a value >= 0 on success or < 0 on failure.

Comments

This function first looks for a built-in command with the specified name. If none is found, it looks for a Stdio application database with
the name "Cmd-cmdname" where cmdname is the first word in the command string cmdStrP.

See Also  SioExecCommand, system

Siovfprintf

Purpose  Write formatted output to the stdout output stream.

Declared In  StdIOPalm.h

Prototype  Int16 Siovfprintf (FILE *fs, const Char *formatP, _Palm_va_list args)

Parameters  
- > fs  An output stream to which to write the formatted output. You can specify only the value stdout for this parameter; alternate streams are not currently implemented.

- > formatP  A pointer to a format string that controls how subsequent arguments are converted for output.

- > args  A pointer to a list of zero or more parameters to be formatted as specified by the formatP string.

Result  Returns the number of characters written out (not including the null terminator used to end output strings). Returns a negative number if there is an error.

Comments  This function internally calls StrVPrintF to do the formatting. See that function for details on which format specifications are supported.

See Also  vfprintf
**sprintf**

**Purpose**  
Macro that calls `StrPrintF` to write formatted output to the stdout output stream.

**Declared In**  
StdIOPalm.h

**Prototype**  
`sprintf (formatP, ...)`

**Parameters**

- `-> formatP`  
  A pointer to a format string that controls how subsequent arguments are converted for output.

- `-> ...`  
  Zero or more parameters to be formatted as specified by the `formatP` string.

**Result**  
Returns the number of characters written out (not including the null terminator used to end output strings).

**Comments**  
See `StrVPrintF` for details on which format specifications are supported. Returns a negative number if there is an error.

**system**

**Purpose**  
Macro that calls `Siosystem` to execute another Stdio command.

**Declared In**  
StdIOPalm.h

**Prototype**  
`system (cmdStrP)`

**Parameters**

- `-> cmdStrP`  
  A pointer to a string containing the command line to execute.

**Result**  
Returns a value >= 0 on success or < 0 on failure.

**Comments**  
This function first looks for a built-in command with the specified name. If none is found, it looks for a Stdio application database with
the name "Cmd-

See Also  

\textbf{vfprintf}

\textbf{Purpose}  

Macro that calls \texttt{Siovfprintf} to write formatted output to the stdout output stream.

\textbf{Declared In}  

\texttt{StdIOPalm.h}

\textbf{Prototype}  

\texttt{vfprintf (fs, formatP, args)}

\textbf{Parameters}  

\begin{itemize}
  \item \texttt{fs}  
  An output stream to which to write the formatted output. You can specify only the value \texttt{stdout} for this parameter; alternate streams are not currently implemented.
  
  \item \texttt{formatP}  
  A pointer to a format string that controls how subsequent arguments are converted for output.
  
  \item \texttt{args}  
  A pointer to a list of zero or more parameters to be formatted as specified by the \texttt{formatP} string.
\end{itemize}

\textbf{Result}  

Returns the number of characters written out (not including the null terminator used to end output strings). Returns a negative number if there is an error.

\textbf{Comments}  

This function internally calls \texttt{StrVPrintF} to do the formatting. See that function for details on which format specifications are supported.
vsprintf

Purpose Macro that calls StrVPrintF to write formatted output to the stdout output stream.

Declared In StdIOPalm.h

Prototype vsprintf (fs, formatP, args)

Parameters

- \(-> \text{fs}\) An output stream to which to write the formatted output. You can specify only the value stdout for this parameter; alternate streams are not currently implemented.

- \(-> \text{formatP}\) A pointer to a format string that controls how subsequent arguments are converted for output.

- \(-> \text{args}\) A pointer to a list of zero or more parameters to be formatted as specified by the \(y\) string.

Result Returns the number of characters written out (not including the null terminator used to end output strings). Returns a negative number if there is an error.

Comments See StrVPrintF for details on which format specifications are supported.

Standard IO Provider Functions

These functions are used by a standard IO provider application.
SioClearScreen

**Purpose**  
Clears the entire standard IO output field.

**Declared In**  
StdIOProvider.h

**Prototype**  
void SioClearScreen (void)

**Parameters**  
None.

**Result**  
Returns nothing.

SioExecCommand

**Purpose**  
Executes a command line.

**Declared In**  
StdIOProvider.h

**Prototype**  
Int16 SioExecCommand (const Char *cmd)

**Parameters**  
-> cmd  
A pointer to a string containing the command line to execute.

**Result**  
Returns a value >= 0 on success or < 0 on failure.

**Comments**  
This function first looks for a built-in command with the specified name. If none is found, it looks for a Stdio application database with the name "Cmd-cmdname" where cmdname is the first word in the command string cmd.

If you pass the string "help" or "?" for the cmd parameter, SioExecCommand causes a help string to be printed for each built-in command. It actually executes each built-in command, passing the string "?" as argv[1]. Each command should handle this argument by printing a help line.

The SioExecCommand function is faster than calling system to execute a command. However, SioExecCommand can be called
Standard IO Provider Functions

only by the standard IO provider application, not the standard IO application.

**SioFree**

**Purpose**
Closes down the standard IO manager.

**Declared In**
StdIOProvider.h

**Prototype**
Err SioFree (void)

**Parameters**
None.

**Result**
Returns 0 on success.

**SioHandleEvent**

**Purpose**
Handles an event in the form that contains the standard IO output field and scroll arrows if the event belongs to the text field or scroll arrows.

**Declared In**
StdIOProvider.h

**Prototype**
Boolean SioHandleEvent (SysEventType *event)

**Parameters**
-> event Pointer to an EventType structure.

**Result**
Returns true if the event was handled and should not be processed by the application’s own form event handler; returns false otherwise.

**Comments**
This function must be called from the form event handler before it does its own processing with any of the objects unrelated to standard IO in the form.
SioInit

**Purpose**  Initializes the standard IO manager.

**Declared In**  StdIOProvider.h

**Prototype**  
```c
Err SioInit (UInt16 formID, UInt16 fieldID, UInt16 scrollerID)
```

**Parameters**
- `formID`  The ID of the form that contains the input/output field.
- `fieldID`  The ID of the field to be used for input/output.
- `scrollerID`  The ID of the scroller associated with the input/output form.

**Result**  Returns 0 on success.
Application-Defined Function

You must supply this function in your stdio application.

**SioMain**

**Purpose**  
The main entry point for the stdio application.

**Declared In**  
StdIOPalm.h

**Prototype**  
Int16 SioMain (UInt16 argc, const Char *argv[])

**Parameters**

- `argc`  
The number of parameters passed on the command line.

- `argv`  
An array of character pointers, one for each parameter passed on the command line.

**Result**  
The return value from this routine is passed back to the system call that invoked it. Return 0 for no error.
String Manager

This chapter provides reference material for the string manager. The string manager API is declared in the header file StringMgr.h.

For more information, see Chapter 8, “Text,” on page 251 of the Palm OS Programmer’s Companion, vol. I.

String Manager Functions

**StrAToI**

**Purpose**
Convert a string to an integer.

**Declared In**
StringMgr.h

**Prototype**
Int32 StrAToI (const Char *str)

**Parameters**
- `str`
  Pointer to a string to convert.

**Result**
Returns the integer.

**Comments**
Use this function instead of the standard `atoi` routine.
StrCaselessCompare

Purpose
Compare two strings with case and accent insensitivity.

Declared In
StringMgr.h

Prototype
Int16 StrCaselessCompare (const Char *s1, const Char *s2)

Parameters
-> s1 Pointer to a string.
-> s2 Pointer to a string.

Result
Returns 0 if the strings match.
Returns a positive number if s1 > s2.
Returns a negative number if s1 < s2.

Comments
Use this function instead of the standard strcmp routine.
To support systems that use multi-byte character encodings, consider using TxtCaselessCompare instead of this function (orTxtCompare for a case-sensitive comparison). Both functions can match single-byte characters with their multi-byte equivalents, butTxtCaselessCompare can also return the length of the matching text.

See Also
StrNCaselessCompare, TxtCaselessCompare, StrCompare, StrNCompare
StrCat

**Purpose**  Concatenate one null-terminated string to another.

**Declared In**  StringMgr.h

**Prototype**  Char *StrCat (Char *dst, const Char *src)

**Parameters**
- `-> dst`  Pointer to the null-terminated destination string.
- `-> src`  Pointer to the null-terminated source string.

**Result**  Returns a pointer to the destination string.

**Comments**  Use this function instead of the standard `strcat` routine.

StrChr

**Purpose**  Look for a character within a string.

**Declared In**  StringMgr.h

**Prototype**  Char *StrChr (const Char *str, WChar chr)

**Parameters**
- `-> str`  Pointer to the string to be searched.
- `-> chr`  Character to search for.

**Result**  Returns a pointer to the first occurrence of character in `str`. Returns `NULL` if the character is not found.

**Comments**  Use this function instead of the standard `strchr` routine.

This function can handle both single-byte characters and multi-byte characters correctly. However, you should make sure that you pass a `WChar` variable to `StrChr` instead of a `Char`. If you pass a `Char` variable, the function sign-extends the variable to a `WChar`, which causes problems if the value is 0x80 or higher.
**String Manager**  
*String Manager Functions*

**Compatibility**
This routine does not correctly find a `\0` character on Palm OS® version 1.0.

**See Also**  
StrStr

**StrCompare**

**Purpose**
Compare two strings.

**Declared In**
StringMgr.h

**Prototype**
Int16 StrCompare (const Char *s1, const Char *s2)

**Parameters**
- `s1` Pointer to a string.
- `s2` Pointer to a string.

**Result**
Returns 0 if the strings match.
Returns a positive number if `s1` sorts after `s2` alphabetically.
Returns a negative number if `s1` sorts before `s2` alphabetically.

**Comments**
Use this function or StrCompareAscii instead of the standard `strcmp` routine. This function is case sensitive.

To support systems that use multi-byte character encodings, consider using TxtCompare instead of this function. Both functions can match single-byte characters with their multi-byte equivalents, butTxtCompare can also return the length of the matching text.

**Compatibility**
Prior to Palm OS 4.0, StrCompare and TxtCompare only performed one level of comparison and returned as soon as they found two unequal characters. For example, if you compared the string “celery” with the string “Cauliflower,” both functions returned a value indicating that “celery” should appear before “Cauliflower” because they sorted “c” before “C.”

In Palm OS 4.0, StrCompare calls TxtCompare, and TxtCompare performs a comparison using up to six comparison tables for sorting.
with increasing precision. As a result, in Palm OS 4.0 and higher, StrCompare sorts “Cauliflower” before “celery.”

See Also  StrNCompare, StrNCaselessCompare, TxtCaselessCompare

StrCompareAscii

Purpose  Compare two ASCII strings.

Declared In  StringMgr.h

Prototype  Int16 StrCompareAscii (const Char *s1, const Char *s2)

Parameters  -> s1 Pointer to a string.
            -> s2 Pointer to a string.

Result  Returns 0 if the strings match.
        Returns a positive number if s1 sorts after s2 alphabetically.
        Returns a negative number if s1 sorts before s2 alphabetically.

Comments  Use this function instead of the standard strcmp routine. Use it to do case-sensitive comparisons on strings that are guaranteed to be 7-bit ASCII strings. This function performs a fast, simple byte-to-byte comparison that is guaranteed never to change.

Compatibility  Implemented only if 4.0 New Feature Set is present.

See Also  StrCompare, StrNCompare, TxtCompare, StrCaselessCompare, StrNCaselessCompare, TxtCaselessCompare, StrNCompareAscii
**String Manager**

*String Manager Functions*

---

### StrCopy

**Purpose**
Copy one string to another.

**Declared In**
StringMgr.h

**Prototype**
Char *StrCopy (Char *dst, const Char *src)

**Parameters**
- `dst` Pointer to the destination string.
- `src` Pointer to the source string.

**Result**
Returns a pointer to the destination string.

**Comments**
Use this function instead of the standard `strcpy` routine.
This function does not work properly with overlapping strings.

---

### StrDelocalizeNumber

**Purpose**
Delocalize a number passed in as a string. Convert the number from any localized notation to US notation (decimal point and thousandth comma). The current thousand and decimal separators have to be passed in.

**Declared In**
StringMgr.h

**Prototype**
Char *StrDelocalizeNumber (Char *s, Char thousandSeparator, Char decimalSeparator)

**Parameters**
- `s` Pointer to the number as an ASCII string.
- `thousandSeparator` Current thousand separator.
- `decimalSeparator` Current decimal separator.

**Result**
Returns a pointer to the changed number and modifies the string in `s`. 
Comments

The current thousandSeparator and decimalSeparator can be determined by obtaining the value of the prefNumberFormat preference using PrefGetPreference and then passing the returned NumberFormatType to LocGetNumberSeparators. For example:

```c
Char *localizedNum;
NumberFormatType numFormat;
Char thousandsSeparator, decimalSeparator;

numFormat = (NumberFormatType) PrefGetPreference(prefNumberFormat);
LocGetNumberSeparators(numFormat, &thousandsSeparator, &decimalSeparator);
StrDelocalizeNumber(localizedNum, thousandsSeparator, decimalSeparator);
```

Compatibility

Implemented only if 2.0 New Feature Set is present.

See Also

StrLocalizeNumber, LocGetNumberSeparators

StrIToA

Purpose

Convert an integer to ASCII.

Declared In

StringMgr.h

Prototype

Char *StrIToA (Char *s, Int32 i)

Parameters

<- s Pointer to a string of size maxStrIToALen in which to store the results.
-> i Integer to convert.

Result

Returns a pointer to the result string.

See Also

StrAToI, StrIToH
StrIToH

**Purpose**
Convert an integer to hexadecimal ASCII.

**Declared In**
StringMgr.h

**Prototype**
Char *StrIToH (Char *s, UInt32 i)

**Parameters**
<s>Pointer to a string in which to store the results.</s>
<i>Integer to convert.</i>

**Result**
Returns the string pointer s.

**See Also**
StrIToA

StrLen

**Purpose**
Compute the length of a string.

**Declared In**
StringMgr.h

**Prototype**
UInt16 StrLen (const Char *src)

**Parameters**
<src>Pointer to a string.</src>

**Result**
Returns the length of the string in bytes.

**Comments**
Use this function instead of the standard strlen routine.

This function returns the length of the string in bytes. On systems that support multi-byte characters, the number returned does not always equal the number of characters.

**Compatibility**
In Palm OS 3.5 and Palm OS 4.x this function was declared to return an Int16. In Palm OS 5, and prior to Palm OS 3.5, this function returns a UInt16.
**StrLocalizeNumber**

**Purpose**  Convert a number (passed in as a string) to localized format, using a specified thousands separator and decimal separator.

**Declared In**  StringMgr.h

**Prototype**  
```c
Char *StrLocalizeNumber (Char *s, 
Char thousandSeparator, Char decimalSeparator)
```
**String Manager**  
*String Manager Functions*

### Parameters

- `<-> s`  
  Numeric ASCII string to localize.

- `-> thousandSeparator`  
  Localized thousand separator.

- `-> decimalSeparator`  
  Localized decimal separator.

### Result

Returns a pointer to the changed number. Converts the number string in `s` by replacing all occurrences of “,” with `thousandSeparator` and all occurrences of “.” with `decimalSeparator`.

### Comments

The current `thousandSeparator` and `decimalSeparator` can be determined by obtaining the value of the `prefNumberFormat` preference using `PrefGetPreference` and then passing the returned `NumberFormatType` to `LocGetNumberSeparators`. For example:

```c
Char *localizedNum;
NumberFormatType numFormat;
Char thousandsSeparator, decimalSeparator;

numFormat = (NumberFormatType)
    PrefGetPreference(prefNumberFormat);
LocGetNumberSeparators(numFormat,
    &thousandsSeparator, &decimalSeparator);
StrLocalizeNumber(localizedNum,
    thousandsSeparator, decimalSeparator);
```

### Compatibility

Implemented only if [2.0 New Feature Set](#) is present.

### See Also

`StrDelocalizeNumber`
**StrNCaselessCompare**

**Purpose**
Compares two strings out to \( n \) characters with case and accent insensitivity.

**Declared In**
StringMgr.h

**Prototype**
```c
Int16 StrNCaselessCompare (const Char *s1, const Char *s2, Int32 n)
```

**Parameters**
- `s1` Pointer to the first string.
- `s2` Pointer to the second string.
- `n` Length in bytes of the text to compare.

**Result**
Returns 0 if the strings match.
Returns a positive number if \( s1 > s2 \).
Returns a negative number if \( s1 < s2 \).

**Comments**
To support systems that use multi-byte character encodings, consider using `TxtCaselessCompare` instead of this function (or `TxtCompare` for a case-sensitive comparison). Both functions can match single-byte characters with their multi-byte equivalents, but `TxtCaselessCompare` can also return the length of the matching text.

**Compatibility**
Implemented only if 2.0 New Feature Set is present. As of Palm OS 4.0, both `s1` and `s2` must be null-terminated strings.

**See Also**
`StrNCompare`, `StrCaselessCompare`, `TxtCaselessCompare`, `StrCompare`
**StrNCat**

**Purpose**  
Concatenates one string to another clipping the destination string to a maximum of \( n \) bytes (including the null character at the end).

---

**IMPORTANT:** The Palm OS implementation of `StrNCat` differs from the implementation in the standard C library. See the Comments section for details.

---

**Declared In**  
`StringMgr.h`

**Prototype**  
```c
Char *StrNCat (Char *dst, const Char *src, Int16 n)
```

**Parameters**  
- `-> dst`  
  Pointer to the null-terminated destination string.
- `-> src`  
  Pointer to the source string.
- `-> n`  
  Maximum length in bytes for `dst`, including the terminating null character.

**Result**  
Returns a pointer to the destination string.

**Comment**  
This function differs from the standard C `strncat` function in these ways:

- `StrNCat` treats the parameter `n` as the maximum length in bytes for `dst`. That means it will copy at most `n - StrLen(dst) - 1` bytes from `src`. The standard C function always copies `n` bytes from `src` into `dst`. (It copies the entire `src` into `dst` if the length of `src` is less than `n`).

- If the length of the destination string reaches `n - 1`, `StrNCat` stops copying bytes from `src` and appends the terminating null character to `dst`. If the length of the destination string is already greater than or equal to `n - 1` before the copying begins, `StrNCat` does not copy any data from `src`.

- In the standard C function, if `src` is less than `n`, the entire `src` string is copied into `dst` and then the remaining space is filled with null characters. `StrNCat` does not fill the
remaining space with null characters in released ROMs. In debug ROMs, `StrNCat` fills the remaining bytes with the value 0xFE.

On systems with multi-byte character encodings, this function makes sure that it does not copy part of a multi-byte character. If the last byte copied from `src` contains the high-order or middle byte of a multi-byte character, `StrNCat` backs up in `dst` until the byte after the end of the previous character, and replaces that byte with a null character.

**Compatibility**

Implemented only if [2.0 New Feature Set](#) is present.

### StrNCompare

**Purpose**

Compare two strings out to `n` bytes. This function is case and accent sensitive.

**Declared In**

`StringMgr.h`

**Prototype**

```c
Int16 StrNCompare (const Char *s1, const Char *s2, Int32 n)
```

**Parameters**

- `-> s1` Pointer to a string.
- `-> s2` Pointer to a string.
- `-> n` Length in bytes of text to compare.

**Result**

Returns 0 if the strings match.

Returns a positive number if `s1 > s2`.

Returns a negative number if `s1 < s2`.

**Comments**

To support systems that use multi-byte character encodings, consider using `TxtCompare` instead of this function. Both functions can match single-byte characters with their multi-byte equivalents, but `TxtCompare` can also return the length of the matching text.
String Manager
String Manager Functions

**Compatibility**
Implemented only if 2.0 New Feature Set is present. As of Palm OS 4.0, both \( s_1 \) and \( s_2 \) must be null-terminated strings.

**See Also**
StrCompare, StrNCaselessCompare, StrCaselessCompare,TxtCaselessCompare, StrCompareAscii

**StrNCompareAscii**

**Purpose**
Compare two ASCII strings out to \( n \) bytes.

**Declared In**
StringMgr.h

**Prototype**
Int16 StrNCompareAscii (const Char *s1, const Char *s2, Int32 n)

**Parameters**
- \( s_1 \) Pointer to a string.
- \( s_2 \) Pointer to a string.
- \( n \) Length in bytes of text to compare.

**Result**
Returns 0 if the strings match.
Returns a positive number if \( s_1 \) sorts after \( s_2 \) alphabetically.
Returns a negative number if \( s_1 \) sorts before \( s_2 \) alphabetically.

**Comments**
Use this function instead of the standard `strcmp` routine. Use it to do case-sensitive comparisons on strings that are guaranteed to be 7-bit ASCII strings. This function performs a fast, simple byte-to-byte comparison that is guaranteed never to change.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
StrCompare, StrNCompare, TxtCompare, StrCaselessCompare, StrNCaselessCompare, TxtCaselessCompare, StrCompareAscii
**StrNCopy**

**Purpose**  Copies up to \( n \) bytes from a source string to the destination string. Terminates \( \text{dst} \) string at index \( n-1 \) if the source string length was \( n-1 \) or less.

**Declared In**  StringMgr.h

**Prototype**  

\[
\text{Char *StrNCopy (Char *dst, const Char *src, Int16 n)}
\]

**Parameters**  

- \( \text{-> dst} \)  Pointer to the destination string.
- \( \text{-> src} \)  Pointer to the source string.
- \( \text{-> n} \)  Maximum number of bytes to copy from \( \text{src} \) string.

**Result**  Returns nothing.

**Comments**  On systems with multi-byte character encodings, this function makes sure that it does not copy part of a multi-byte character. If the \( n \)th byte of \( \text{src} \) contains the high-order or middle byte of a multi-byte character, \text{StrNCopy} backs up in \( \text{dst} \) until the byte after the end of the previous character, and replaces the remaining bytes (up to \( n-1 \)) with nulls.

Be aware that the \( n \)th byte of \( \text{dst} \) upon return may contain the last byte of a multi-byte character. If you plan to terminate the string by setting its last character to NULL, you must not pass the entire length of the string to \text{StrNCopy}. If you do, your code may overwrite the final byte of the last character.

```c
// WRONG! You may overwrite part of multi-byte character.
Char dst[n];
StrNCopy(dst, src, n);
dst[n-1] = chrNull;
```

Instead, if you write to the last byte of the destination string, pass one less than the size of the string to \text{StrNCopy}. 
// RIGHT. Instead pass n-1 to StrNCopy.
Char dst[n];
StrNCopy(dst, src, n-1);
dst[n-1] = chrNull;

Compatibility Implemented only if 2.0 New Feature Set is present.

StrPrintF

Purpose Implements a subset of the ANSI C sprintf call, which writes formatted output to a string.

Declared In StringMgr.h

Prototype Int16 StrPrintF (Char *s,
const Char *formatStr, ...)

Parameters -> s Pointer to a string into which the results are written.
-> formatStr Pointer to the format specification string.
... Zero or more arguments to be formatted as specified by formatStr.

Result Number of characters written to destination string. Returns a negative number if there is an error.

Comments This function internally calls StrVPrintF to do the formatting. See that function for details on which format specifications are supported.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also StrVPrintF
StrStr

Purpose    Look for a substring within a string.

Declared In  StringMgr.h

Prototype  Char *StrStr (const Char *str, const Char *token)

Parameters
- > str  Pointer to the string to be searched.
- > token  Pointer to the string to search for.

Result  Returns a pointer to the first occurrence of token in str or NULL if not found.

Comments  Use this function instead of the standard strstr routine.

On systems with multi-byte character encodings, this function makes sure that it does not match only part of a multi-byte character. If the matching strings begins at an inter-character boundary, then this function returns NULL.

NOTE: If the value of the token parameter is the empty string, this function returns NULL. This is different than the standard strstr function, which returns str when token is the empty string.

See Also  StrChr

StrToLower

Purpose    Convert all the characters in a string to lowercase.

Declared In  StringMgr.h

Prototype  Char *StrToLower (Char *dst, const Char *src)

Parameters
- > dst  Pointer to a string.
String Manager
String Manager Functions

-> src Pointer to a null-terminated string.

Result Returns a pointer to the destination string.

Compatibility Prior to Palm OS version 3.5, this function only converted accented characters on Japanese devices. On Palm OS version 3.5 and higher, all characters are appropriately lowercased, including accented characters on Latin devices.

StrVPrintF

Purpose Implements a subset of the ANSI C vsprintf call, which writes formatted output to a string.

Declared In StringMgr.h

Prototype Int16 StrVPrintF (Char *s, const Char *formatStr, va_list arg)

Parameters
<- s Pointer to a string into which the results are written. This string is always terminated by a null terminator.

-> formatStr Pointer to the format specification string.

-> arg Pointer to a list of zero or more parameters to be formatted as specified by the formatStr string.

Result Number of characters written to destination string, not including the null terminator. Returns a negative number if there is an error.

Comments Like the C vsprintf function, this function is designed to be called by your own function that takes a variable number of arguments and passes them to this function. For details on how to use it, see “Using the StrVPrintF Function” on page 267 of the Palm OS Programmer’s Companion, vol. I, or refer to vsprintf in a standard C reference book.
Currently, only the conversion specifications `%d`, `%i`, `%u`, `%x`, `%s`, and `%c` are implemented by `StrVPrintF` (and related functions). Optional modifiers that are supported include: `+`, `-`, `<space>`, `*`, `<digits>`, `h` and `l` (long). Following is a brief description of how these format specifications work (see a C book for more details).

Each conversion specification begins with the `%` character. Following the `%` character, there may be one or more of the characters list in Table 47.1, in sequence.

**Table 47.1 StrVPrintF Format Specification**

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
</table>
| +         | Specifies that a sign always be placed before a number produced by a signed conversion. A + overrides a space if both are used. Example: 
  `StrPrintF(s, "%+d  +%d", 4, -5);`  
  Output to `s`:  
  +4   -5 |
| -         | Specifies that the printed value is left justified within the field width allowed for it. Example:  
  `StrPrintF(s, "%5d%5d%5d", 6, 7);`  
  Output to `s`:  
  69  78 |
| `<space>` | Specifies that a minus sign always be placed before a negative number and a space before a positive number. Example: 
  `StrPrintF(s, "% d  % d", 4, -5);`  
  Output to `s`:  
  4   -5 |
**Table 47.1 StrVPrintF Format Specification (continued)**

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
</table>
| *         | Indicates that the next argument (must be an integer) in the list specifies the field width. In this case, the argument following that one is used for the value of this field. Example: 
StrPrintF(s,"%*d%d",4,8,5);
Output to s: 8 5 |
| <number>  | Specifies a minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces on the left (or right, if the left justified flag is also specified) to fill out the field width. Example: 
StrPrintF(s,"%d%5d",4,3);
Output to s: 4 3 |
| h         | Specifies that the following d, i, u, or x conversion corresponds to a short or unsigned short argument. Example: 
StrPrintF(s,"%hd",401);
Output to s: 401 |
| l or L    | Specifies that the following d, i, u, x, or c conversion corresponds to a long or unsigned long 
StrPrintF(s,"%ld",999999999);
Output to s: 999999999 |
| <character> | A character that indicates the type of conversion to be performed. The supported conversion characters include: |
Table 47.1 StrVPrintF Format Specification (continued)

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>d or i</td>
<td>A signed integer argument is converted to decimal notation. Example: \texttt{StrPrintF(s, &quot;%d %d&quot;, 4, -4)}; Output to \texttt{s}: 4 -4</td>
</tr>
<tr>
<td>u</td>
<td>An unsigned integer argument is converted to decimal notation. Example: \texttt{StrPrintF(s, &quot;%u %u&quot;, 4, -4)}; Output to \texttt{s}: 4 65532</td>
</tr>
<tr>
<td>x</td>
<td>An integer argument is converted to hexadecimal notation. Example: \texttt{StrPrintF(s, &quot;%x&quot;, 125)}; Output to \texttt{s}: 0000007D</td>
</tr>
<tr>
<td>s</td>
<td>A string (char *) argument is copied to the destination string. Example: \texttt{StrPrintF(s, &quot;ABC%s&quot;, &quot;DEF&quot;)}; Output to \texttt{s}: ABCDEF</td>
</tr>
<tr>
<td>c or C</td>
<td>A single character argument is copied to the destination string. If \texttt{C} is used or if the \texttt{l} modifier is used, the argument must be a WChar. Example: \texttt{StrPrintF(s, &quot;Telephone%c&quot;, 's')}; Output to \texttt{s}: Telephones</td>
</tr>
<tr>
<td>%</td>
<td>A % character is copied to the destination string. Example: \texttt{StrPrintF(s, &quot;%&quot;)}; Output to \texttt{s}: %</td>
</tr>
</tbody>
</table>

Example

Here’s an example of how to use this call:
#include <unix_stdio.h>
void MyPrintf(Char *s, Char *formatStr, ...)
{
    va_list args;
    Char text[0x100];
    va_start(args, formatStr);
    StrVPrintF(text, formatStr, args);
    va_end(args);
    MyPutS(text);
}

**Compatibility**  Implemented only if [2.0 New Feature Set](#) is present.

**See Also**  [StrPrintF](#)
System Event Manager

This chapter describes functions available in the system event manager. The system event manager API is declared in the header files Event.h and SysEvtMgr.h.

For more information on the system event manager, see the chapter “Event Loop” in the Palm OS Programmer’s Companion, vol. I. The reference for specific events sent by the system are documented in “Palm OS Events.”

System Event Manager Data Structures

The following system event manager data structures are documented in the “Palm OS Events” chapter:

- `eventsEnum`
- `EventType`
- `EventPtr`
System Event Manager Functions

EvtAddEventToQueue

Purpose: Add an event to the event queue.

Declared In: Event.h

Prototype: void EvtAddEventToQueue (const EventType *event)

Parameters:
- `event`: Pointer to the structure that contains the event.

Result: Returns nothing.

Comments: This function makes a copy of the structure that you pass in and adds it to the event queue.

EvtAddUniqueEventToQueue

Purpose: Add an event to the event queue, replacing one of the same type if it is found.

Declared In: Event.h

Prototype: void EvtAddUniqueEventToQueue (const EventType *eventP, UInt32 id, Boolean inPlace)

Parameters:
- `eventP`: Pointer to the structure that contains the event.
- `id`: ID of the event. 0 means match only on the type.
- `inPlace`: If true, any existing event is replaced. If false, the existing event is deleted and a new event is added to end of queue.

Result: Returns nothing.
Comments  This function looks for an event in the event queue of the same event type and ID (if specified). The routine replaces it with the new event, if found.

If no existing event is found, the new event is copied to the queue.

If an existing event is found, the routine proceeds as follows:

• If `inPlace` is `true`, the existing event is replaced with a copy of the new event.
• If `inPlace` is `false`, the existing event is removed and the new event is added to the end of the queue.

Compatibility  Implemented only if [2.0 New Feature Set](#) is present.
EvtCopyEvent

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Copy an event.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declared In</td>
<td>Event.h</td>
</tr>
<tr>
<td>Prototype</td>
<td>void EvtCopyEvent (const EventType *source, EventType *dest)</td>
</tr>
<tr>
<td>Parameters</td>
<td>-&gt; source Pointer to the structure containing the event to copy.</td>
</tr>
<tr>
<td></td>
<td>&lt;- dest Pointer to the structure to copy the event to.</td>
</tr>
<tr>
<td>Result</td>
<td>Returns nothing.</td>
</tr>
<tr>
<td>Comments</td>
<td>Use this function only if you want to create an event that has the same type as the source event. The data field in an EventType structure is specific to events of a given type. If you were to use this function to copy a keyDownEvent and then change it to a frmLoadEvent, the resulting frmLoadEvent would not have the proper data field.</td>
</tr>
<tr>
<td></td>
<td>If you want to create an event of a different type, do not use EvtCopyEvent. First clear the EventType structure using MemSet and then change the event type:</td>
</tr>
<tr>
<td></td>
<td>MemSet(&amp;event, sizeof(EventType), 0);</td>
</tr>
<tr>
<td></td>
<td>event.eType = frmLoadEvent;</td>
</tr>
<tr>
<td></td>
<td>event.data.frmLoad.formID = formID;</td>
</tr>
<tr>
<td></td>
<td>EvtAddEventToQueue(&amp;event);</td>
</tr>
</tbody>
</table>
EvtDequeuePenPoint

Purpose   Get the next pen point out of the pen queue. This function is called by recognizers.

Declared In  SysEvtMgr.h

Prototype  Err EvtDequeuePenPoint (PointType *retP)

Parameters  <- retP  Return point.

Result  Always returns 0.

Comments  Called by a recognizer that wishes to extract the points of a stroke. Returns the point (-1, -1) at the end of a stroke.

Before calling this routine, you must call EvtDequeuePenStrokeInfo.

EvtDequeuePenStrokeInfo

Purpose   Initiate the extraction of a stroke from the pen queue.

Declared In  SysEvtMgr.h

Prototype  Err EvtDequeuePenStrokeInfo (PointType *startPtP, PointType *endPtP)

Parameters  <- startPtP  Start point returned here.
<- endPtP  End point returned here.

Result  Always returns 0.

Comments  Called by the system function EvtGetSysEvent when a penUpEvent is being generated. This routine must be called before EvtDequeuePenPoint is called.

Subsequent calls to EvtDequeuePenPoint return points at the starting point in the stroke and including the end point. After the
end point is returned, the next call to `EvtDequeuePenPoint` returns the point -1, -1.

See Also  `EvtDequeuePenPoint`

**EvtEnableGraffiti**

**Purpose** Set Graffiti® enabled or disabled.

**Declared In** SysEvtMgr.h

**Prototype** void EvtEnableGraffiti (Boolean enable)

**Parameters**
- `-> enable` true to enable Graffiti, false to disable Graffiti.

**Result** Returns nothing.

**EvtEnqueueKey**

**Purpose** Place keys into the key queue.

**Declared In** SysEvtMgr.h

**Prototype** Err EvtEnqueueKey (WChar ascii, UInt16 keycode, UInt16 modifiers)

**Parameters**
- `-> ascii` Character code for the key.
- `-> keycode` Virtual key code of key. This is the keyCode field of the `keyDownEvent` and is currently unused.
- `-> modifiers` Modifiers for `keyDownEvent`.

**Result** Returns 0 if successful, or `evtErrParamErr` if an error occurs.
Comments  This function disables interrupts while the queue header is being modified because both interrupt- and non-interrupt-level code can post keys into the queue.

Entries in the key queue only take 1 byte if the ascii parameter has a value less than 256 and the keycode and modifiers parameters are both zero. Otherwise an entry can take up to 7 bytes.

IMPORTANT:  Make sure you pass a WChar as the ascii parameter, not a Char. If you pass a high-ASCII Char, the compiler sign-extends it to be a 16-bit value, resulting in the wrong character being added to the key queue.

EvtEventAvail

Purpose  Return true if an event is available.

Declared In  Event.h

Prototype  Boolean EvtEventAvail (void)

Parameters  None.

Result  Returns true if an event is available, false otherwise.

Compatibility  Implemented only if 2.0 New Feature Set is present.
**EvtFlushKeyQueue**

**Purpose**
Flush all keys out of the key queue.

**Declared In**
SysEvtMgr.h

**Prototype**
Err EvtFlushKeyQueue (void)

**Parameters**
None.

**Result**
Always returns 0.

**Comments**
Called by the system function EvtSetPenQueuePtr.

**EvtFlushNextPenStroke**

**Purpose**
Flush the next stroke out of the pen queue.

**Declared In**
SysEvtMgr.h

**Prototype**
Err EvtFlushNextPenStroke ()

**Parameters**
None.

**Result**
Always returns 0.

**Comments**
Called by recognizers that need only the start and end points of a stroke. If a stroke has already been partially dequeued (by EvtDequeuePenStrokeInfo) this routine finishes the stroke dequeueing. Otherwise, this routine flushes the next stroke in the queue.

**See Also**
EvtDequeuePenPoint
EvtFlushPenQueue

Purpose        Flush all points out of the pen queue.

Declared In   SysEvtMgr.h

Prototype     Err EvtFlushPenQueue (void)

Parameters    None

Result        Always returns 0.

Comments      Called by the system function EvtSetKeyQueuePtr.

See Also      EvtPenQueueSize

EvtGetEvent

Purpose        Return the next available event.

Declared In   Event.h

Prototype     void EvtGetEvent (EventType *event, Int32 timeout)

Parameters  <- event    Pointer to the structure to hold the event returned.
             -> timeout  Maximum number of ticks to wait before an event is returned (evtWaitForever means wait indefinitely).

Comments      Pass evtWaitForever as the timeout in most instances. When running on the device, this makes the CPU go into doze mode until the user provides input. For applications that do animation, pass a timeout value greater than or equal to zero.

              Note that a timeout value greater than or equal to zero is simply the maximum number of ticks which can elapse before EvtGetEvent returns an event. If any other event—including a nilEvent—
occurs before this time has elapsed, \texttt{EvtGetEvent} will return that event. Otherwise, once the specified time has elapsed \texttt{EvtGetEvent} generates and returns a \texttt{nilEvent}. If you supply a value of zero for the timeout parameter, \texttt{EvtGetEvent} returns the event currently in the queue, or, if there aren't any events in the queue, it immediately generates and returns a \texttt{nilEvent}.

**Result**

Returns nothing.

### EvtGetPen

**Purpose**

Return the current status of the pen.

**Declared In**

\texttt{Event.h}

**Prototype**

\begin{verbatim}
void EvtGetPen (Int16 *pScreenX, Int16 *pScreenY, Boolean *pPenDown)
\end{verbatim}

**Parameters**

\begin{itemize}
  \item \texttt{pScreenX} \quad x location relative to display.
  \item \texttt{pScreenY} \quad y location relative to display.
  \item \texttt{pPenDown} \quad true or false.
\end{itemize}

**Result**

Returns nothing.

**Comments**

Called by various UI routines.

**See Also**

\texttt{EvtGetPenNative}, \texttt{KeyCurrentState}
EvtGetPenBtnList

**Purpose**
Return a pointer to the silk-screen button array.

**Declared In**
SysEvtMgr.h

**Prototype**
```
const PenBtnInfoType *EvtGetPenBtnList
(UInt16 *numButtons)
```

**Parameters**
- `numButtons`<br>The number of elements in the returned array.

**Result**
Returns a pointer to an array of the silk-screen buttons.

**Comments**
This function returns an array of `PenBtnInfoType` structures:
```
typedef struct PenBtnInfoType {
    RectangleType boundsR;
    WCHAR asciiCode;
    UINT16 keyCode;
    UINT16 modifiers;
} PenBtnInfoType;
```
The fields in the `PenBtnInfoType` contain the following information:

- `boundsR`<br>The button’s bounding rectangle.
- `asciiCode`<br>The character code generated when the button is tapped. This is typically a virtual character.
- `keyCode`<br>Currently unused.
- `modifiers`<br>Modifiers for the key down event. (See the description of the modifiers field for `keyDownEvent`.)

The number of buttons is device-dependent. Even devices with the same Palm OS® version may have differing numbers of silk-screen buttons. For example, Japanese devices typically have four extra
silk-screen buttons used to transliterate characters into different alphabets.

See Also  EvtProcessSoftKeyStroke

New  EvtGetPenNative

Purpose  Get the current status of the pen using a window’s active coordinate system.

Declared In  Window.h

Prototype  void EvtGetPenNative (WinHandle winH, Int16 *pScreenX, Int16 *pScreenY, Boolean *pPenDown)

Parameters  
-> winH  Handle to a valid window.
<- pScreenX  x location relative to the window.
<- pScreenY  y location relative to the window.
<- pPenDown  true if the pen is down, false otherwise.

Result  Returns nothing.

Comments  This function is a variation on EvtGetPen. EvtGetPen returns a pen sample using the standard coordinate system, relative to the draw window, whereas EvtGetPenNative returns a pen sample using the active coordinate system of winH, relative to the window origin. If the active coordinate system is high density, the returned pen sample uses high-density coordinates.

On a debug ROM this function displays an error if winH doesn’t reference a valid window object.

Compatibility  Implemented only if the High-Density Display Feature Set is present.
EvtGetSilkscreenAreaList

**Purpose**  Returns a pointer to the silk-screen area array. This array contains the bounds of each silk-screen area.

**Declared In**  SysEvtMgr.h

**Prototype**  
```c
const SilkscreenAreaType *
EvtGetSilkscreenAreaList (UInt16 *numAreas)
```

**Parameters**  
`numAreas`  The number of elements in the returned array.

**Result**  Returns a pointer to an array containing the bounds of each silk-screen area.

**Comments**  This function returns an array of the SilkscreenAreaType structures:

```c
typedef struct SilkscreenAreaType {
    RectangleType bounds;
    UInt32    areaType;
    UInt16    index;
} SilkscreenAreaType;
```

The fields in this structure provide the following information.

`bounds`  The area’s bounds.
System Event Manager
System Event Manager Functions

areaType  The area type, can be one of the following:
  silkscreenRectGraffiti  The Graffiti area.
  silkscreenRectScreen  The entire silkscreen area.

Depending on the handheld manufacturer, Palm Powered™ devices may have other area types.

index  If the area type is silkscreenRectGraffiti, this field is either alphaGraffitiSilkscreenArea to represent the portion where letters are entered or numericGraffitiSilkscreenArea to represent the portion where numbers are entered.

Compatibility  Implemented only if 3.5 New Feature Set is present. If 5.0 New Feature Set is present, this function should be considered “System Use Only”; applications should do what they can to avoid using it.

EvtKeydownIsVirtual

Purpose  Macro that indicates if eventP is a pointer to a virtual character key down event.

Declared In  Event.h

Prototype  EvtKeydownIsVirtual (eventP)

Parameters  -> eventP  Pointer to an EventType structure.

Result  Returns true if the character is a letter in an alphabet or a numeric digit, false otherwise.

Comments  The macro assumes that the caller has already determined the event is a keyDownEvent.

This macro is intended for use by the system. Applications should use TxtGlueCharIsVirtual, contained in the PalmOSGlue Library.
Compatibility  Implemented in the Palm OS 3.5 SDK.

See Also  TxtGlueCharIsVirtual

**EvtKeyQueueEmpty**

**Purpose**  Return true if the key queue is currently empty.

**Declared In**  SysEvtMgr.h

**Prototype**  Boolean EvtKeyQueueEmpty (void)

**Parameters**  None.

**Result**  Returns true if the key queue is currently empty, otherwise returns false.

**Comments**  Usually called by the key manager to determine if it should enqueue auto-repeat keys.

**EvtKeyQueueSize**

**Purpose**  Return the size of the current key queue in bytes.

**Declared In**  SysEvtMgr.h

**Prototype**  UInt32 EvtKeyQueueSize (void)

**Parameters**  None.

**Result**  Returns size of queue in bytes.

**Comments**  Called by applications that wish to see how large the current key queue is.
**EvtPenQueueSize**

**Purpose**
Return the size of the current pen queue in bytes.

**Declared In**
SysEvtMgr.h

**Prototype**
UInt32 EvtPenQueueSize (void)

**Parameters**
None.

**Result**
Returns size of queue in bytes.

**Comments**
Call this function to see how large the current pen queue is.

**EvtProcessSoftKeyStroke**

**Purpose**
Translate a stroke in the system area of the digitizer and enqueue the appropriate key events into the key queue.

**Declared In**
SysEvtMgr.h

**Prototype**
Err EvtProcessSoftKeyStroke (PointType *startPtP, PointType *endPtP)

**Parameters**
- >startPtP Start point of stroke.
- >endPtP End point of stroke.

**Result**
Returns 0 if recognized, -1 if not recognized.

**See Also**
EvtGetPenBtnList, GrfProcessStroke
EvtResetAutoOffTimer

Purpose  Reset the auto-off timer.

Declared In  SysEvtMgr.h

Prototype  Err EvtResetAutoOffTimer (void)

Parameters  None.

Result  Always returns 0.

Comments  Called by the serial link manager; can be called periodically by other managers.

EvtResetAutoOffTimer resets the auto-off timer so that the device does not turn off until at least the default amount of idle time has passed. You can use this function to ensure that the device doesn’t automatically power off during a long operation without user input (for example, when there is a lot of serial port activity).

If you need more control over the auto-off timer and the 3.5 New Feature Set is present, consider using EvtSetAutoOffTimer instead of this function.

See Also  SysSetAutoOffTime

EvtSetAutoOffTimer

Purpose  Set the auto-off timer.

Declared In  SysEvtMgr.h

Prototype  Err EvtSetAutoOffTimer (EvtSetAutoOffCmd cmd, UInt16 timeout)

Parameters  -> cmd  One of the defined commands.
System Event Manager
System Event Manager Functions

-A new timeout value in seconds. If cmd is ResetTimer, this parameter is ignored.

Result Always returns errNone.

Comments Use EvtSetAutoOffTimer to ensure that the device doesn’t automatically power off during a long operation that has no user input (for example, when there is a lot of serial port activity). The cmd parameter specifies the operation you want to perform. It takes one of the following EvtSetAutoOffCmd constants:

SetAtLeast Make sure that the device won’t turn off until timeout seconds of idle time has passed. (This operation only changes the current value if it’s less than the value you specify.)

SetExactly Set the timer to turn off in timeout seconds

SetAtMost Make sure the device will turn before timeout seconds has passed. (This operation only changes the current value if it’s greater than the value you specify.)

SetDefault Change the default auto-off timeout to timeout seconds.

ResetTimer Reset the auto-off timer so that the device does not turn off until at least the default seconds of idle time has passed.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also EvtResetAutoOffTimer, SysSetAutoOffTime
EvtSetNullEventTick

**Purpose**
Make sure a nilEvent occurs in at least the specified number of ticks.

**Declared In**
SysEvtMgr.h

**Prototype**
Boolean EvtSetNullEventTick(UInt32 tick)

**Parameters**
- `tick` Maximum number of system ticks that should elapse before a nilEvent is added to the queue.

**Result**
Returns true if timeout value changed, or false if it did not change.

**Compatibility**
In versions prior to Palm OS 3.5, this function was implemented as a macro.

EvtSysEventAvail

**Purpose**
Return true if a low-level system event (such as a pen or key event) is available.

**Declared In**
SysEvtMgr.h

**Prototype**
Boolean EvtSysEventAvail(Boolean ignorePenUps)

**Parameters**
- `ignorePenUps` If true, this routine ignores pen-up events when determining if there are any system events available.

**Result**
Returns true if a system event is available.

**Comment**
Call EvtEventAvail to determine whether high-level software events are available.

**Compatibility**
Implemented only if 2.0 New Feature Set is present.
**EvtWakeup**

**Purpose**
Force the event manager to wake up and send a `nilEvent` to the current application.

**Declared In**
SysEvtMgr.h

**Prototype**
`Err EvtWakeup (void)`

**Parameters**
None.

**Result**
Always returns 0.

**Comments**
Called by interrupt routines, like the sound manager and alarm manager.

**See Also**
EvtWakeupWithoutNilEvent

**EvtWakeupWithoutNilEvent**

**Purpose**
Force the event manager to wake up without sending a `nilEvent` to the current application.

**Declared In**
SysEvtMgr.h

**Prototype**
`Err EvtWakeupWithoutNilEvent()`

**Parameters**
None.

**Result**
Always returns 0.

**Comments**
Called by interrupt routines.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
EvtWakeup
System Manager

This chapter provides reference material for the system manager. The system manager API is declared in the header files SystemMgr.h and SysUtils.h.

For more information on the system manager, see the chapters “Application Startup and Stop” and “Palm System Support” in the Palm OS Programmer’s Companion, vol. I.

System Manager Data Structures

SysDBListItemType

The SysDBListItemType structure describes a single database or panel. The SysCreateDataBaseList and SysCreatePanelList functions each create and return an array of SysDBListItemType structures.

typedef struct {
    Char         name[dmDBNameLength];
    UInt32       creator;
    UInt32       type;
    UInt16       version;
    LocalID      dbID;
    UInt16       cardNo;
    BitmapPtr    iconP;
} SysDBListItemType;
System Functions

SysAppLaunch

**Purpose**
Launch a specified application as a subroutine of the caller.

**Declared In**
SystemMgr.h

**Prototype**
```
Err SysAppLaunch (UInt16 cardNo, LocalID dbID, UInt16 launchFlags, UInt16 cmd, MemPtr cmdPBP, UInt32 *resultP)
```

**Parameters**
- `-> cardNo, dbID` The card number and ID of the resource database of the application to launch.
- `-> launchFlags` Set to 0.
- `-> cmd` Launch code.
- `-> cmdPBP` Launch code parameter block.
- `<- resultP` The value returned from the application’s [PilotMain](#) routine.

**Result**
Returns 0 if no error, or one of `sysErrParamErr`, `memErrNotEnoughSpace`, or `sysErrOutOfOwnerIDs`.

**Comments**
Applications can use `SysAppLaunch` to send a specific launch code to another application and have control return to the calling application when finished. This function in effect makes the specified application a subroutine of the caller. If you want to actually close your application and call another application, use [SysUIAppSwitch](#) instead of this function. `SysUIAppSwitch` sends the current application an `appStopEvent` and then starts the specified application.

Do not use this function to open the system-supplied Application Launcher application. If another application has replaced the default launcher with one of its own, this function will open the custom launcher instead of the system-supplied one. To open the system-supplied launcher reliably, enqueue a `keyDownEvent` that
contains a launchChr, as shown in the section “Application Launcher” of the user interface chapter in the Palm OS Programmer’s Companion, vol. I.

NOTE: For important information regarding the correct use of this function, see the “Application Startup and Stop” chapter in the Palm OS Programmer’s Companion, vol. I.

See Also  
SysBroadcastActionCode, SysUIAppSwitch, SysCurAppDatabase

**SysBatteryInfo**

**Purpose**  
Retrieve settings for the batteries. Set `set` to `false` to retrieve battery settings. (Applications should not change any of the settings).

**WARNING!** Use this function only to retrieve settings!

**Declared In**  
SystemMgr.h

**Prototype**  
UInt16 SysBatteryInfo (Boolean set, UInt16 *warnThresholdP, UInt16 *criticalThresholdP, Int16 *maxTicksP, SysBatteryKind *kindP, Boolean *pluggedIn, UInt8 *percentP)

**Parameters**

- **set**  
  If `false`, parameters with non-NULL pointers are retrieved. Never set this parameter to `true`.

- **warnThresholdP**  
  Pointer to battery voltage warning threshold in volts*100, or NULL.

- **criticalThresholdP**  
  Pointer to the battery voltage critical threshold in volts*100, or NULL.

- **maxTicksP**  
  Pointer to the battery timeout, or NULL.
kindP Pointer to the battery kind, or NULL.
pluggedIn Pointer to pluggedIn return value, or NULL.
percentP Percentage of power remaining in the battery.

Result Returns the current battery voltage in volts*100.

Comments Call this function to make sure an upcoming activity won’t be interrupted by a low battery warning.

warnThresholdP and maxTicksP are the battery-warning voltage threshold and time out. If the battery voltage falls below the threshold, or the timeout expires, a lowBatteryChr key event is put on the queue. Normally, applications callSysHandleEvent which calls SysBatteryDialog in response to this event.

criticalThresholdP is the battery voltage threshold. If battery voltage falls below this level, the system turns itself off without warning and doesn’t turn on until battery voltage is above it again.

Compatibility This function was revised for Palm OS® 3.0. In Palm OS 3.0, the percentP parameter was added. This enhancement is implemented only if 3.0 New Feature Set is present.

See Also SysBatteryInfoV20

SysBatteryInfoV20

Purpose Retrieve settings for the batteries. Set to false to retrieve battery settings. (Applications should not change any of the settings).

WARNING! Use this function only to retrieve settings!
Declared In
SystemMgr.h

Prototype
UInt16 SysBatteryInfoV20 (Boolean set,
UInt16 *warnThresholdP,
UInt16 *criticalThresholdP, Int16 *maxTicksP,
SysBatteryKind *kindP, Boolean *pluggedIn)

Parameters
set  
If false, parameters with non-NULL pointers are retrieved. Never set this parameter to true.

warnThresholdP  Pointer to battery voltage warning threshold in volts*100, or NULL.

criticalThresholdP  Pointer to the battery voltage critical threshold in volts*100, or NULL.

maxTicksP  Pointer to the battery timeout, or NULL.

kindP  Pointer to the battery kind, or NULL.

pluggedIn  Pointer to pluggedIn return value, or NULL.

Result
Returns the current battery voltage in volts*100.

Comments
Call this function to make sure an upcoming activity won’t be interrupted by a low battery warning.

warnThresholdP and maxTicksP are the battery-warning voltage threshold and time out. If the battery voltage falls below the threshold, or the timeout expires, a lowBatteryChr key event is put on the queue. Normally, applications call SysHandleEvent which calls SysBatteryDialog in response to this event.

criticalThresholdP is the battery voltage threshold. If battery voltage falls below this level, the system turns itself off without warning and doesn’t turn on until battery voltage is above it again.

Compatibility
This function corresponds to the Palm OS 2.0 version of SysBatteryInfo. Implemented only if 3.0 New Feature Set is present.

See Also  SysBatteryInfo
SysBinarySearch

Purpose  Search elements in a sorted array for the specified data according to the specified comparison function.

Declared In  SysUtils.h

Prototype  

Boolean SysBinarySearch (void const *baseP, Int16 numOfElements, Int16 width, SearchFuncPtr searchF, void const *searchData, Int32 other, Int32 *position, Boolean findFirst)
Parameters  
- baseP: Base pointer to an array of elements
- numOfElements: Number of elements to search. Must be greater than 0.
- width: Width of each array element.
- searchF: Search function.
- searchData: Data to search for. This data is passed to the searchF function.
- other: Data to be passed as the third parameter (the other parameter) to the comparison function.
- position: Pointer to the position result.
- findFirst: If set to true, the first matching element is returned. Use this parameter if the array contains duplicate entries to ensure that the first such entry will be the one returned.

Result  
- Returns true if an exact match was found. In this case, position points to the element number where the data was found.
- Returns false if an exact match was not found. If false is returned, position points to the element number where the data should be inserted if it was to be added to the array in sorted order.

Comments  
The array must be sorted in ascending order prior to the search. Use SysInsertionSort or SysQSort to sort the array.

The search starts at element 0 and ends at element (numOfElements - 1).

The search function’s (searchF) prototype is:

\[
\text{Int16 } \_\text{searchF (void const *searchData, void const *arrayData, Int32 other);}\]

The first parameter is the data for which to search, the second parameter is a pointer to an element in the array, and the third parameter is any other necessary data.
The function returns:

- $>0$ if the search data is greater than the element
- $<0$ if the search data is less than the element
- $0$ if the search data is the same as the element

**Compatibility**

Implemented only if [2.0 New Feature Set](#) is present.

**SysBroadcastActionCode**

**Purpose**

Send the specified action code (launch code) and parameter block to the latest version of every UI application.

**Declared In**

`SystemMgr.h`

**Prototype**

```
Err SysBroadcastActionCode (UInt16 cmd, MemPtr cmdPBP)
```

**Parameters**

- `cmd` Action code to send.
- `cmdPBP` Action code parameter block to send.

**Result**

Returns 0 if no error, or one of the following errors: `sysErrParamErr`, `memErrNotEnoughSpace`, or `sysErrOutOfOwnerIDs`.

**See Also**

`SysAppLaunch`, Chapter 2, “Application Startup and Stop.” of the *Palm OS Programmer’s Companion*, vol. I
SysCopyStringResource

**Purpose**  
Copy a resource string to a passed string.

**Declared In**  
SysUtils.h

**Prototype**  
```c
void SysCopyStringResource (Char *string, Int16 theID)
```

**Parameters**
- **string**  
  String to copy the resource string to.
- **theID**  
  Resource string ID.

**Result**  
Stores a copy of the resource string in string.
SysCreateDataBaseList

Purpose
Generate a list of databases found on the memory cards matching a specific type and return the result. If lookupName is true then a name in a tAIN resource is used instead of the database’s name and the list is sorted. Only the last version of a database is returned. Databases with multiple versions are listed only once.

Declared In
SystemMgr.h

Prototype
Boolean SysCreateDataBaseList (UInt32 type, UInt32 creator, UInt16 *dbCount, MemHandle *dbIDs, Boolean lookupName)

Parameters
- type
  The type of database to find. Use 0 to find all databases.

- creator
  The creator ID of the database to find. Use 0 to find any creator ID.

- dbCount
  A pointer to an integer value that is updated by this function to the number of matching databases.

- dbIDs
  A pointer to a handle that gets allocated to contain the database list. Upon return, this references an array of SysDBListItemType structures. See the Comments section below for more information.

- lookupName
  If true, SysCreateDatabaseList uses tAIN names and sorts the list.

Result
Returns false if no databases were found, and true if any databases were found. The value of dbCount is updated to reflect the number of databases that were found. If at least one database is found, dbIDs is updated to reference a array of SysDBListItemType structures; this array contains dbCount items.
Comments

This function creates a list of unique databases, where unique is defined as having a different type and creator ID. Two or more databases with the same type and creator ID are counted as one. Thus, you cannot use `SysCreateDatabaseList` to build a list of databases that share a common type and creator. There are two exceptions to this rule, however. If `type` is 0 or if `creator` is not 0, the code that removes “non-unique” databases isn’t run. It also isn’t run if the type is `sysFileTpqa`, since web-clipping databases all have the same type and creator ID.

If this function returns `true` and the value of `dbCount` is greater than 0, than you can iterate through the list of database items, as shown in Listing 49.1.

Listing 49.1 Using the `SysCreateDatabaseList` function

```c
SysDBListItemType *dbListIDsP;
MemHandle        dbListIDsH;
UInt16           dbCount = 0;
Boolean          status;
UInt16           counter;
SysDBListItemType theItem;

status = SysCreateDatabaseList(sysFileTpqa, 0,
                                   &dbCount, &dbListIDsH, false);

if (status == true && dbCount > 0)
{
    dbListIDsP = MemHandleLock (dbListIDsH);
    for (counter = 0; counter < dbCount; counter++)
        if StrCompare(dbListIDsP[counter].name, "MINE") == 0
            // we found my database
            ...
            ...
    MemPtrFree (dbListIDsP);
}
```

**NOTE:** It is your responsibility to free the memory allocated by this function for the list of databases.

Compatibility

Implemented only if 2.0 New Feature Set is present.
SysCreatePanelList

**Purpose**
Generate a list of panels found on the memory cards and return the result. Multiple versions of a panel are listed once.

**Declared In**
SystemMgr.h

**Prototype**
Boolean SysCreatePanelList (UInt16 *panelCount, MemHandle *panelIDs)

**Parameters**
- **panelCount**
  Pointer to set to the number of panels.
- **panelIDs**
  A pointer to a handle that gets allocated to contain the panel list. Upon return, this references an array of **SysDBListItemType** structures.

**Result**
Returns false if no panels were found, and true if any panels were found. The value of panelCount is updated to reflect the number of panels that were found. If at least one panel is found, panelIDs is updated to reference a array of **SysDBListItemType** structures; this array contains panelCount items.

**Comments**
If this function returns true and the value of panelCount is greater than 0, than you can iterate through the list of panel items, as shown in **Listing 49.1**. It is your responsibility to free the memory allocated for the panel list.

**Compatibility**
Implemented only if 2.0 New Feature Set is present.
SysCurAppDatabase

**Purpose**  Return the card number and database ID of the current application’s resource database.

**Declared In**  SystemMgr.h

**Prototype**  Err SysCurAppDatabase (UInt16 *cardNoP, LocalID *dbIDP)

**Parameters**  
- cardNoP  Pointer to the card number; 0 or 1.
- dbIDP  Pointer to the database ID.

**Result**  Returns 0 if no error, or SysErrParamErr  if an error occurs.

**See Also**  SysAppLaunch, SysUIAppSwitch

SysErrString

**Purpose**  Returns text to describe an error number. This routine looks up the textual description of a system error number in the appropriate List resource and creates a string that can be used to display that error.

The actual string will be of the form: "<error message> (XXXX)" where XXXX is the hexadecimal error number.

This routine looks for a resource of type 'tstl' and resource ID of (err>>8). It then grabs the string at index (err & 0x00FF) out of that resource.

The first string in the resource is called index #1 by Constructor, NOT #0. For example, an error code of 0x0101 will fetch the first string in the resource.

**Declared In**  SysUtils.h

**Prototype**  Char *SysErrString (Err err, Char *strP, UInt16 maxLen)

**Parameters**  
- err  Error number
**SysFormPointerArrayToStrings**

**Purpose**
Form an array of pointers to strings in a block. Useful for setting the items of a list.

**Declared In**
SysUtils.h

**Prototype**
MemHandle SysFormPointerArrayToStrings (Char *c, Int16 stringCount)

**Parameters**
- `c` Pointer to packed block of strings, each terminated by a null character.
- `stringCount` Count of strings in block.

**Result**
Unlocked handle to allocated array of pointers to the strings in the passed block. The returned array points to the strings in the passed packed block. Note that you’ll need to free the returned handle when you no longer need it.

**See Also**
[LstSetListChoices](#)
**New**

sysFtrNumProcessorIs68K

**Purpose**
Macro that determines whether or not the underlying processor is part of the 68K family.

**Declared In**
SystemMgr.h

**Prototype**
```
#define sysFtrNumProcessorIs68K(x)
    (((x&sysFtrNumProcessor68KIfZero)==0)? true : false)
```

**Parameters**
-> x  
Processor type obtained from a call to `FtrGet`.

**Result**
Returns true if the underlying processor is a 68K, false otherwise.

**Comments**
This macro is typically used in conjunction with `PceNativeCall`.

**Example**
```
UInt32 processorType;
FtrGet(sysFileCSystem, sysFtrNumProcessorID, &processorType);
if (sysFtrNumProcessorIs68K(processorType)){
    // processor is 68K
} else {
    // processor is not 68K
}
```
New sysFtrNumProcessorIsARM

**Purpose**
Macro that determines whether or not the underlying processor is part of the ARM family.

**Declared In**
SystemMgr.h

**Prototype**
```
define sysFtrNumProcessorIsARM(x)
  ((x&sysFtrNumProcessorARMIfNotZero)!=0)? true : false
```

**Parameters**
- `x` Processor type obtained from a call to FtrGet.

**Result**
Returns true if the underlying processor is an ARM core, false otherwise.

**Comments**
This macro is typically used in conjunction with PceNativeCall.

**Example**
```
UInt32 processorType;

FtrGet(sysFileCSystem, sysFtrNumProcessorID, &processorType);
if (sysFtrNumProcessorIsARM(processorType))
  // processor is ARM
} else {
  // processor is not ARM
}
SysGetOSVersionString

Purpose  Return the version number of the Palm™ operating system.

Declared In  SystemMgr.h

Prototype  Char *SysGetOSVersionString()

Parameters  None.

Result  Returns a string such as “v. 3.0.”

Comments  You must free the returned string using the MemPtrFree function.

Compatibility  Implemented only if 3.0 New Feature Set is present.

SysGetROMToken

Purpose  Return from ROM a value specified by token.

Declared In  SystemMgr.h

Prototype  Err SysGetROMToken (UInt16 cardNo, UInt32 token, 
UInt8 **dataP, UInt16 *sizeP)

Parameters  
-> cardNo  The card on which the ROM to be queried resides. Currently, no Palm hardware provides multiple cards, so this value must be 0.

-> token  The value to retrieve, as specified by one of the following tokens:

sysROMTokenSnum  The serial number of the ROM, expressed as a text string with no null terminator.

<- dataP  Pointer to a text buffer that holds the requested value when the function returns.
System Manager
System Functions

sizeP

The number of bytes in the dataP buffer.

Result

Returns the requested value if no error, or an error code if an error occurs. If this function returns an error, or if the returned pointer to the buffer is NULL, or if the first byte of the text buffer is 0xFF, then no serial number is available.

Comments

The serial number is shown to the user in the Application Launcher, along with a checksum digit you can use to validate input when your users read the ID from their device and type it in or tell it to someone else.

Compatibility

Implemented only if 3.0 New Feature Set is present. Serial numbers are available only on flash ROM-based units.

See Also

“Retrieving the ROM Serial Number” section in the Palm OS Programmer’s Companion, vol. I shows how to retrieve the ROM serial number and calculate its associated checksum.

SysGetStackInfo

Purpose

Return the start and end of the current thread’s stack.

Declared In

SystemMgr.h

Prototype

Boolean SysGetStackInfo (MemPtr *startPP, MemPtr *endPP)

Parameters

startPP

Upon return, points to the start of the stack.

endPP

Upon return, points to the end of the stack.

Result

Returns true if the stack has not overflowed, that is, the value of the stack overflow address has not been changed. Returns false if the stack overflow value has been overwritten, meaning that a stack overflow has occurred.

Compatibility

Implemented only if 3.0 New Feature Set is present.
SysGetTrapAddress

Purpose  Return the address of a function given its system trap.

Declared In  SystemMgr.h

Prototype  void *SysGetTrapAddress (UInt16 trapNum)

Parameters  -> trapNum  One of the routine selectors defined in 
               SysTraps.h (sysTrap...) or CoreTraps.h on Palm OS version 3.5 and higher.

Result  Returns the address of the corresponding function. Returns the 
        address of the sysTrapSysUnimplemented routine if an invalid 
        routine selector is passed; compare the results of this function to 
        SysGetTrapAddress(sysTrapSysUnimplemented) if you 
        need to check for an invalid routine selector.

Comments  Use this function for performance reasons. You can then use the 
           address it returns to call the function without having to go through 
           the trap dispatch table. This function is mostly useful for optimizing 
           the performance of functions called in a tight loop.

           The Palm OS trap dispatch mechanism allows the trap table entries 
           to be modified at any time, either as the result of a system update or 
           a hack. For this reason, it’s important to call this function 
           immediately before entering the tight loop. If the trap address 
           changes in between when you call SysGetTrapAddress and you 
           use the address, the wrong function will be called.

Compatibility  On Palm OS 3.0 and earlier, this function contains a bug that causes 
                it to return a garbage value if an invalid routine selector is passed. 
                To prevent this bug from affecting your application, use 
                SysGlueGetTrapAddress in the PalmOSGlue library instead of 
                calling this function directly. For more information, see Chapter 75,
                “PalmOSGlue Library.”
SysGremlins

**Purpose**
Query the Gremlins facility. You pass a selector for a function and parameters for that function. Gremlins performs the function call and returns the result.

**Declared In**
SysUtils.h

**Prototype**
```c
UInt32 SysGremlins (GremlinFunctionType selector,
                     GremlinParamsType *params)
```

**Parameters**
- **selector**
  The selector for a function to pass to Gremlins.
- **params**
  Pointer to a parameter block used to pass parameters to the function specified by selector.

**Result**
Returns the result of the function performed in Gremlins.

**Comments**
Currently, only one selector is defined, GremlinIsOn, which takes no parameters. GremlinIsOn returns 0 if Gremlins is not running, non-zero if it is running.

Currently, non-zero values are returned only from the version of Gremlins in the Palm OS emulator. The Gremlins running in the Simulator on a Macintosh and over the serial line via the Palm Debugger return zero for GremlinIsOn.

Use this function if you need to alter the application’s behavior when Gremlins is running. For example, the debug 3.0 ROM refuses to bring up the digitizer panel when Gremlins is running under the emulator.

**Compatibility**
Implemented only if [3.0 New Feature Set](#) is present.

In Palm OS 3.2 and later, SysGremlins is replaced by the functions defined in the file HostControl.h. Specifically, the one selector supported by SysGremlins is replaced with the function HostGremlinIsRunning. For backward compatibility, SysGremlins is mapped to HostGremlinIsRunning.
SysHandleEvent

**Purpose**  Handle defaults for system events such as hard and soft key presses.

**Declared In**  SystemMgr.h

**Prototype**  Boolean SysHandleEvent (EventPtr eventP)

**Parameters**  
- **eventP**  Pointer to an event.

**Result**  Returns true if the system handled the event.

**Comments**  Applications should call this routine immediately after calling EvtGetEvent unless they want to override the default system behavior. However, overriding the default system behavior is almost never appropriate for an application.

**See Also**  EvtProcessSoftKeyStroke, KeyRates

SysInsertionSort

**Purpose**  Sort elements in an array according to the passed comparison function.

**Declared In**  SysUtils.h

**Prototype**  
```c
void SysInsertionSort (void *baseP, Int16 numOfElements, Int16 width, CmpFuncPtr comparF, Int32 other)
```

**Parameters**  
- **baseP**  Base pointer to an array of elements.
- **numOfElements**  Number of elements to sort (must be at least 2).
- **width**  Width of an element.
- **comparF**  Comparison function. See Comments, below.
other  Other data passed to the comparison function.

Result  Returns nothing.
Comments

Only elements which are out of order move. Moved elements are moved to the end of the range of equal elements. If a large amount of elements are being sorted, try to use the quick sort (see `SysQSort`).

This is the insertion sort algorithm: Starting with the second element, each element is compared to the preceding element. Each element not greater than the last is inserted into sorted position within those already sorted. A binary search for the insertion point is performed. A moved element is inserted after any other equal elements.

In order to use `SysInsertionSort` you must write a comparison function with the following prototype:

```c
Int16 comparF (void *p1, void *p2, Int32 other)
```

Your comparison function must return zero if `p1` equals `p2`, a positive integer value if `p1` is greater than `p2`, and a negative integer value if `p1` is less than `p2`. Note that the value of the parameter named `other` is passed through from the `SysInsertionSort` call and can be used to control the behavior of the `comparF` function if appropriate for your application.

Compatibility

The Palm OS 2.0 comparison function (`comparF`) has this prototype:

```c
Int comparF (VoidPtr, VoidPtr, Long other);
```

The Palm OS 1.0 comparison function (`comparF`) has this prototype:

```c
Int comparF (BytePtr A, BytePtr B, Long other);
```

See Also

`SysQSort`
**SysKeyboardDialog**

**Purpose**
Pop up the system keyboard if there is a field object with the focus. The field object’s text chunk is edited directly.

**Declared In**
Keyboard.h

**Prototype**
void SysKeyboardDialog (KeyboardType kbd)

**Parameters**
kbd
The keyboard type. See Keyboard.h.

**Result**
Returns nothing. Changes the field’s text chunk.

**Compatibility**
Implemented only if 2.0 New Feature Set is present.

**See Also**
SysKeyboardDialogV10, FrmSetFocus

**SysKeyboardDialogV10**

**Purpose**
Pop up the system keyboard if there is a field object with the focus. The field object’s text chunk is edited directly.

**Declared In**
Keyboard.h

**Prototype**
void SysKeyboardDialogV10 ()

**Parameters**
None.

**Result**
Returns nothing. The field’s text chunk is changed.

**Compatibility**
Corresponds to the 1.0 implementation of SysKeyboardDialog.

**See Also**
SysKeyboardDialog, FrmSetFocus
SysLibFind

Purpose
Return a reference number for a library that is already loaded, given its name.

Declared In
SystemMgr.h

Prototype
Err SysLibFind (const Char *nameP, UInt16 *refNumP)

Parameters
nameP Pointer to the name of a loaded library.
refNumP Pointer to a variable for returning the library reference number (on failure, this variable is undefined)

Result
0 if no error; otherwise: sysErrLibNotFound (if the library is not yet loaded), or another error returned from the library’s install entry point.

Comments
Most built-in libraries (NetLib, serial, IR) are preloaded automatically when the system is reset. Third-party libraries must be loaded before this call can succeed (use SysLibLoad). You can check if a library is already loaded by calling SysLibFind and checking for a 0 error return value (it will return a non-zero value if the library is not loaded).

SysLibInstall

Purpose
Installs a library into the Library table and calls the library’s install entry point.

Declared In
SystemMgr.h

Prototype
Err SysLibInstall (SysLibEntryProcPtr libraryP, UInt16 *refNumP)

Parameters
-> libraryP Pointer to the library being installed.
<- refNumP  Pointer to the variable in which the installed library reference number is stored upon return.

Result  Returns 0 if no error, memErrNotEnoughSpace if an error occurs while allocating memory in the library table, or, if an error is returned by the call to the library’s install entry point, that error is returned and the reference number is set to sysInvalidRefNum.

Comments  This routine is largely used by the Palm OS, but can also be called by applications that need to install their own libraries.

Compatibility  Implemented only if 2.0 New Feature Set is present.

SysLibLoad

Purpose  Load a library given its database creator and type.

Declared In  SystemMgr.h

Prototype  Err SysLibLoad (UInt32 libType, UInt32 libCreator, UInt16 *refNumP)

Parameters  
- libType  Type of library database.
- libCreator  Creator of library database.
- refNumP  Pointer to variable for returning the library reference number (on failure, sysInvalidRefNum is returned in this variable)

Result  0 if no error; otherwise: sysErrLibNotFound, sysErrNoFreeRAM, sysErrNoFreeLibSlots, or other error returned from the library’s install entry point.

Comments  Presently, the “load” functionality is not supported when you use the Palm OS Simulator.

When an application no longer needs a library that it successfully loaded via SysLibLoad, it is responsible for unloading the library
by calling `SysLibRemove` and passing it the library reference number returned by `SysLibLoad`. More information is available in the white paper on shared libraries, which you can find on the Palm developer support web site.

**Compatibility**

Implemented only if [2.0 New Feature Set](#) is present.

### SysLibRemove

**Purpose**

Unload a library previously loaded with `SysLibLoad`.

**Declared In**

`SystemMgr.h`

**Prototype**

```c
Err SysLibRemove (UInt16 refNum)
```

**Parameters**

-> `refNum` The library reference number.

**Result**

0 if no error; otherwise `sysErrParamErr` if the `refNum` is not a valid library reference number.

**Comments**

`SysLibRemove` releases the dynamic memory allocated to the shared library’s dispatch table, resources, and global variables.

### SysQSort

**Purpose**

Sort elements in an array according to the supplied comparison function.

**Declared In**

`SysUtils.h`

**Prototype**

```c
void SysQSort (void *baseP, Int16 numOfElements, Int16 width, CmpFuncPtr comparF, Int32 other)
```

**Parameters**

- `baseP` Base pointer to an array of elements.
- `numOfElements` Number of elements to sort (must be at least 2).
- `width` Width of an element.
System Manager
System Functions

comparF  Comparison function. See Comments, below.
other  Other data passed to the comparison function.

Result  Returns nothing.

Comments  Equal records can be in any position relative to each other because a quick sort tends to scramble the ordering of records. As a result, calling SysQSort multiple times can result in a different order if the records are not completely unique. If you don’t want this behavior, use the insertion sort instead (see SysInsertionSort).

To pick the pivot point, the quick sort algorithm picks the middle of three records picked from around the middle of all records. That way, the algorithm can take advantage of partially sorted data.

These optimizations are built in:

• The routine contains its own stack to limit uncontrolled recursion. When the stack is full, an insertion sort is used because it doesn’t require more stack space.

• An insertion sort is also used when the number of records is low. This avoids the overhead of a quick sort which is noticeable for small numbers of records.

• If the records seem mostly sorted, an insertion sort is performed to move only those few records that need to be moved.

In order to use SysQSort you must write a comparison function with the following prototype:

    Int16 comparF (void *p1, void *p2, Int32 other)

Your comparison function must return zero if p1 equals p2, a positive integer value if p1 is greater than p2, and a negative integer value if p1 is less than p2. Note that the value of the parameter named other is passed through from the SysQSort call and can be used to control the behavior of the comparF function if appropriate for your application.

Compatibility  The Palm OS 2.0 comparison function (comparF) has this prototype:
Int comparF (VoidPtr, VoidPtr, Long other);

The Palm OS 1.0 comparison function (comparF) has this prototype:

Int comparF (BytePtr A, BytePtr B, Long other);

See Also  SysInsertionSort

SysRandom

Purpose  Return a random number anywhere from 0 to sysRandomMax.

Declared In  SysUtils.h

Prototype  Int16 SysRandom (Int32 newSeed)

Parameters  newSeed  New seed value, or 0 to use existing seed.

Result  Returns a random number.

SysReset

Purpose  Perform a soft reset and reinitialize the globals and the dynamic memory heap.

Declared In  SystemMgr.h

Prototype  void SysReset (void)

Parameters  None.

Result  No return value.

Comments  This routine resets the system, reinitializes the globals area and all system managers, and reinitializes the dynamic heap. All database information is preserved. This routine is called when the user presses the hidden reset switch on the device.
When running an application using the simulator, this routine looks for two data files that represent the memory of card 0 and card 1. If these are found, the Palm OS memory image is created using them. If they are not found, they are created.

When running an application on the device, this routine simply looks for the memory cards at fixed locations.

**SysSetAutoOffTime**

**Purpose**
Set the time out value in seconds for auto-power-off. Zero means never power off.

**Declared In**
SystemMgr.h

**Prototype**
UInt16 SysSetAutoOffTime (UInt16 seconds)

**Parameters**
- **seconds**
  Time out in seconds, or 0 for no time out.

**Result**
Returns previous value of time out in seconds.

**SysSetTrapAddress**

**Purpose**
Set the address of the function corresponding to a system trap.

**Declared In**
SystemMgr.h

**Prototype**
Err SysSetTrapAddress (UInt16 trapNum, void *procP)

**Parameters**
- **-> trapNum**
  One of the routine selectors defined in SysTraps.h (sysTrap...) or CoreTraps.h on Palm OS version 3.5 and higher.
- **-> procP**
  Pointer to a function that the trap identified by trapNum is to point to.

**Result**
Returns 0 if no error, or SysErrParamErr if trapNum is greater than the number of traps in the trap table.
Comments

This function is useful for patching a system trap, in combination with `SysGetTrapAddress`. To patch a system trap in your application, first call `SysGetTrapAddress` to get the trap address and then save this value somewhere. Next use `SysSetTrapAddress` to set the trap address to point to your function. Before your application exits, remove the patch by calling `SysSetTrapAddress` and passing in the original trap address you saved.

**WARNING!** If your application patches a system trap using this function, you **must** remove the patch before your application exits. Do **not** use this mechanism to permanently patch system traps as it may cause unpredictable results for the system and other applications.

Compatibility

If [5.0 New Feature Set](#) is present this function is unimplemented.

**SysStringByIndex**

**Purpose**

Copy a string out of a string list resource by index. String list resources are of type `tSTL` and contain a list of strings and a prefix string.

ResEdit always displays the items in the list as starting at 1, not 0. Consider this when creating your string list.

**Declared In**

`SysUtils.h`

**Prototype**

```c
Char *SysStringByIndex (UInt16 resID, UInt16 index, Char *strP, UInt16 maxLen)
```

**Parameters**

- `resID`: Resource ID of the string list.
- `index`: String to get out of the list.
- `strP`: Pointer to space to form the string.
SysTaskDelay

**Purpose**  Put the processor into doze mode for the specified number of ticks.

**Declared In**  SystemMgr.h

**Prototype**  Err SysTaskDelay (Int32 delay)

**Parameters**
- delay  Number of ticks to wait (see SysTicksPerSecond)

**Result**  Returns 0 if no error.

**See Also**  EvtGetEvent

SysTicksPerSecond

**Purpose**  Return the number of ticks per second. This routine allows applications to be tolerant of changes to the ticks per second rate in the system.

**Declared In**  SystemMgr.h

**Prototype**  UInt16 SysTicksPerSecond (void)

**Parameters**  None

**Result**  Returns the number of ticks per second.
### Compatibility
Implemented only if [2.0 New Feature Set](#) is present.

### SysUIAppSwitch

**Purpose**  
Try to make the current UI application quit and then launch the UI application specified by card number and database ID.

**Declared In**  
SystemMgr.h

**Prototype**  
Err SysUIAppSwitch (UInt16 cardNo, LocalID dbID, UInt16 cmd, MemPtr cmdPBP)
System Manager
System Functions

Parameters

- **cardNo**: Card number for the new application; currently only card 0 is valid.
- **dbID**: ID of the new application’s resource database.
- **cmd**: Action code (launch code).
- **cmdPBP**: Action code (launch code) parameter block.

Result

Returns 0 if no error.

May display a fatal error message if the **cardNo** parameter is invalid. On debug ROMs, displays a fatal error message if there is no currently running application.

Comments

Do not use this function to open the system-supplied Application Launcher application. If a third-party launch is installed, you’ll likely want to launch that one instead. To do this, enqueue a **keyDownEvent** that contains a **launchChr**, as shown in the section “Application Launcher” of the user interface chapter in the *Palm OS Programmer’s Companion*, vol. I. This will run whatever is run whenever you tap on the Applications icon.

If you are passing a parameter block (the **cmdPBP** parameter), you must set the owner of the parameter block chunk to the operating system. To do this, and for more information, see **MemPtrSetOwner**. If the parameter block structure contains references by pointer or handle to any other chunks, you also must set the owner of those chunks by using **MemHandleSetOwner** or **MemPtrSetOwner**. If you set the owner of this parameter block properly, the system maintains the parameter block and frees it when the second application quits. If you don’t set the owner of the parameter block, the system frees the parameter block as soon as the calling application quits, causing unpredictable results.

See Also

Application-Defined Functions

PilotMain

Purpose
The entry point for all Palm OS applications, this function’s sole purpose is to receive and respond to launch codes.

Declared In
SystemMgr.h

Prototype
UInt32 PilotMain(UInt16 cmd, void *cmdPBP, UInt16 launchFlags)

Parameters
-> cmd
The launch code to which your application is to respond. See Chapter 1, “Application Launch Codes,” on page 3 for a list of predefined launch codes. You may create additional launch codes; see “Creating Your Own Launch Codes” on page 28 of the Palm OS Programmer’s Companion, vol. I.

-> cmdPBP
A pointer to a structure containing any launch-command-specific parameters, or NULL if the launch code has none. See the description of each launch code for a description of the parameter structure that accompanies it, if any.

-> launchFlags
Flags that indicate whether your application’s global variables are available, whether your application is now the active application, whether it already was the active application, and so on. See “Launch Flags” on page 36 for a list of launch flags.

Result
Return errNone if your application processed the launch code successfully, or an appropriate error code if there was a problem. When another application invokes your application using SysAppLaunch, this value is returned to the caller.
Text Manager

This chapter provides information about the text manager API declared in TextMgr.h by discussing these topics:

- Text Manager Data Structures
- Text Manager Functions

For more information on the text manager, see the chapter “Localized Applications” on page 363 in the Palm OS Programmer’s Companion, vol. I.

Text Manager Data Structures

CharEncodingType

The CharEncodingType enum specifies possible character encodings. The Character Encoding Constants define the possible values for CharEncodingType variables.

    UInt8 CharEncodingType;

A given device supports a single character encoding. The currently available devices support either the Palm™ version of Windows code page 1252 (an extension of ISO Latin 1) or the Palm version of Windows code page 932 (an extension of Shift JIS). These encodings are identical to their Windows counterparts with some additional characters added in the control range.

Compatibility

Prior to version 4.0, CharEncodingType was an enum that defined only eight character encodings. The Palm OS® 4.0 definition of CharEncodingType is compatible with the previous definition.
Text Manager Functions

TxtByteAttr

**Purpose**
Return the possible locations of a given byte within a multi-byte character.

**Declared In**
TextMgr.h

**Prototype**
UInt8 TxtByteAttr (UInt8 inByte)

**Parameters**
-> inByte A byte representing all or part of a valid character.

**Result**
Returns a byte with one or more of the following bits set:
- byteAttrFirst First byte of multi-byte character.
- byteAttrLast Last byte of multi-byte character.
- byteAttrMiddle Middle byte of multi-byte character.

**Comments**
If inByte is valid in more than one location of a character, multiple return bits are set. For example, 0x40 in the Shift JIS character encoding is valid as a single-byte character and as the low-order byte of a double-byte character. Thus, the return value for TxtByteAttr(0x40) on a Shift JIS system has both the byteAttrSingle and byteAttrLast bits set.

Text manager functions that need to determine the byte positioning of a character use TxtByteAttr to do so. You rarely need to use this function yourself.

**Compatibility**
Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueByteAttr. For more information, see Chapter 75, “PalmOSGlue Library.”
**TxtCaselessCompare**

**Purpose** Perform a case-insensitive comparison of two text buffers.

**Declared In** TextMgr.h

**Prototype**

```c
Int16 TxtCaselessCompare (const Char* s1, UInt16 s1Len, UInt16* s1MatchLen, const Char* s2, UInt16 s2Len, UInt16* s2MatchLen)
```

**Parameters**

- **-> s1** Pointer to the first text buffer to compare.
- **-> s1Len** Length in bytes of the text pointed to by s1.
- **<- s1MatchLen** Points to the offset of the first character in s1 that determines the sort order. Pass NULL for this parameter if you don’t need to know this number.
- **-> s2** Pointer to the second text buffer to compare.
- **-> s2Len** Length in bytes of the text pointed to by s2.
- **<- s2MatchLen** Points to the offset of the first character in s2 that determines the sort order. Pass NULL for this parameter if you don’t need to know this number.

**Result** Returns one of the following values:

- **< 0** If s1 occurs before s2 in alphabetical order.
- **> 0** If s1 occurs after s2 in alphabetical order.
- **0** If the two substrings that were compared are equal.

**Comments** In certain character encodings (such as Shift JIS), one character may be accurately represented as either a single-byte character or a multi-byte character. `TxtCaselessCompare` accurately matches a single-byte character with its multi-byte equivalent. For this reason, the values returned in s1MatchLen and s2MatchLen are not always equal.
You must make sure that the parameters s1 and s2 point to a the start of a valid character. That is, they must point to the first byte of a multi-byte character or they must point to a single-byte character; if they don’t, results are unpredictable.

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueCaselessCompare. For more information, see Chapter 75, “PalmOSGlue Library.”

In Palm OS 4.0, the TxtCaselessCompare function terminates when it finds a null byte in the string. In earlier releases, it terminated only when it reached the ending byte specified by the length parameters.

See Also StrCaselessCompare, TxtCompare, StrCompare

TxtCharAttr

Purpose Return a character’s attributes.

Declared In TextMgr.h

Prototype UInt16 TxtCharAttr (WChar inChar)

Parameters -> inChar Any valid character.

Result Returns a 16-bit unsigned value with any of the following bits set:

- charAttrPrint Printable
- charAttrSpace Blank space, tab, or newline
- charAttrAlNum Alphanumeric
- charAttrAlpha Alphabetic
- charAttrCtrl Control character
The character passed to this function must be a valid character given the system encoding.

This function is used in the text manager’s character attribute macros (TxtCharIsAlNum, TxtCharIsCntrl, and so on). The macros perform operations analogous to the standard C functions isPunct, isPrintable, and so on. Usually, you’d use one of these macros instead of calling TxtCharAttr directly.

To obtain attributes specific to a given character encoding, use TxtCharXAttr.

Compatibility

Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueCharAttr. For more information, see Chapter 75, “PalmOSGlue Library.”

See Also
TxtCharIsValid

TxtCharBounds

Purpose
Return the boundaries of a character containing the byte at a specified offset in a string.

Declared In
TextMgr.h

Prototype
WChar TxtCharBounds (const Char* inText, UInt32 inOffset, UInt32* outStart, UInt32* outEnd)

Parameters
-> inText Pointer to the text buffer to search.
-> inOffset A valid offset into the buffer inText. This location may contain a byte in any position (start, middle, or end) of a multi-byte character.
Text Manager
Text Manager Functions

<- outStart  Points to the starting offset of the character containing the byte at inOffset.
<- outEnd    Points to the ending offset of the character containing the byte at inOffset.

Result  Returns the character located between the offsets outStart and outEnd.

Comments  Use this function to determine the boundaries of a character in a string or text buffer.

If the byte at inOffset is valid in more than one location of a character, the function must search back toward the beginning of the text buffer until it finds an unambiguous byte to determine the appropriate boundaries. For this reason, TxtCharBounds is often slow and should be used only where needed.

You must make sure that the parameter inText points to the beginning of the string. That is, if the string begins with a multi-byte character, inText must point to the first byte of that character; if it doesn’t, results are unpredictable.

Compatibility  Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueCharBounds. For more information, see Chapter 75, “PalmOSGlue Library.”
**TxtCharEncoding**

**Purpose**  
Return the minimum encoding required to represent a character.

**Declared In**  
TextMgr.h

**Prototype**  
CharEncodingType TxtCharEncoding (WChar inChar)

**Parameters**  
-> inChar  
A valid character.

**Result**  
A CharEncodingType value that indicates the minimum encoding required to represent inChar. If the character isn’t recognizable, charEncodingUnknown is returned.

**Comments**  
The minimum encoding is the encoding that takes the lowest number of bytes to represent the character. For example, if the character is a blank or a tab character, the minimum encoding is charEncodingAscii because these characters can be represented in single-byte ASCII. If the character is a ü, the minimum encoding is charEncodingISO8859_1.

Because Palm OS only supports a single character encoding at a time, the result of this function is always logically equal to or less than the encoding used on the current system. That is, you’ll only receive a return value of charEncodingISO8859_1 if you’re running on a US or European system and you pass a non-ASCII character.

Use this function for informational purposes only. Your code should not assume that the character encoding returned by this function is the Palm OS system character encoding. (Instead use FtrGet as shown in the TxtCharXAttr function description.)

Use **TxtMaxEncoding** to determine the order of encodings.

**Compatibility**  
Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call
TxtGlueCharEncoding. For more information, see Chapter 75, "PalmOSGlue Library."

See Also  TxtStrEncoding, TxtMaxEncoding

**TxtCharsAINum**

**Purpose**  Macro that indicates if the character is alphanumeric.

**Declared In**  TxtMgr.h

**Prototype**  TxtCharIsAINum (ch)

**Parameters**  -> ch  A valid character.

**Result**  Returns true if the character is a letter in an alphabet or a numeric digit, false otherwise.

**Compatibility**  Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueCharIsAINum. For more information, see Chapter 75, "PalmOSGlue Library."

See Also  TxtCharIsDigit, TxtCharIsAlpha

**TxtCharsAlpha**

**Purpose**  Macro that indicates if a character is a letter in an alphabet.

**Declared In**  TxtMgr.h

**Prototype**  TxtCharIsAlpha (ch)

**Parameters**  -> ch  A valid character.

**Result**  Returns true if the character is a letter in an alphabet, false otherwise.
TxtCharIsCntrl

Purpose
Macro that indicates if a character is a control character.

Declared In
TxtMgr.h

Prototype
TxtCharIsCntrl (ch)

Parameters
-> ch  
A valid character.

Result
Returns true if the character is a non-printable character, such as the bell character or a carriage return; false otherwise.

Compatibility
Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueCharIsCntrl. For more information, see Chapter 75, “PalmOSGlue Library.”

TxtCharIsDelim

Purpose
Macro that indicates if a character is a delimiter.

Declared In
TxtMgr.h

Prototype
TxtCharIsDelim (ch)

Parameters
-> ch  
A valid character.

Result
Returns true if the character is a word delimiter (whitespace or punctuation), false otherwise.

Compatibility
Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueCharIsCntrl. For more information, see Chapter 75, “PalmOSGlue Library.”
TxtCharIsDigit

**Purpose**
Macro that indicates if the character is a decimal digit.

**Declared In**
TxtMgr.h

**Prototype**
TxtCharIsDigit (ch)

**Parameters**
- ch 
  A valid character.

**Result**
Returns true if the character is 0 through 9, false otherwise.

**Compatibility**
Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueCharIsDigit. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**
TxtCharIsAlNum, TxtCharIsHex

TxtCharIsGraph

**Purpose**
Macro that indicates if a character is a graphic character.

**Declared In**
TxtMgr.h

**Prototype**
TxtCharIsGraph (ch)

**Parameters**
- ch 
  A valid character.

**Result**
Returns true if the character is a graphic character, false otherwise.
Comments  A graphic character is any character visible on the screen, in other words, letters, digits, and punctuation marks. A blank space is not a graphic character because it is not visible.

This macro differs from `TxtCharIsPrint` in that it returns `false` if the character is whitespace. `TxtCharIsPrint` returns `true` if the character is whitespace.

Compatibility  Valid only if `International Feature Set` is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharIsGraph`. For more information, see Chapter 75, “PalmOSGlue Library.”

**TxtCharIsHardKey**

Purpose  Macro that returns `true` if the character is one of the hard keys on the device.

Declared In  `TxtMgr.h`

Prototype  `TxtCharIsHardKey (m, ch)`

Parameters  
- `<m>` The modifier keys from the `keyDownEvent`.
- `<ch>` The character from the `keyDownEvent`.

Result  `true` if the character is one of the built-in hard keys on the device, `false` otherwise.

Compatibility  Valid only if `International Feature Set` is present.

See Also  `ChrIsHardKey`
TpTextManager

Text Manager Functions

**TxtCharIsHex**

**Purpose**
Macro that indicates if a character is a hexadecimal digit.

**Declared In**
TxtMgr.h

**Prototype**
TxtCharIsHex (ch)

**Parameters**
-> ch  A valid character.

**Result**
Returns true if the character is a hexadecimal digit from 0 to F, false otherwise.

**Compatibility**
Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueCharIsHex. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**
TxtCharIsDigit, TxtCharIsLower

**TxtCharIsLower**

**Purpose**
Macro that indicates if a character is a lowercase letter.

**Declared In**
TxtMgr.h

**Prototype**
TxtCharIsLower (ch)

**Parameters**
-> ch  A valid character.

**Result**
Returns true if the character is a lowercase letter, false otherwise.

**Compatibility**
Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueCharIsLower. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**
TxtCharIsAlpha, TxtCharIsUpper
**TxtCharIsPrint**

**Purpose**  Macro that indicates if a character is printable.

**Declared In**  TxtMgr.h

**Prototype**  TxtCharIsPrint (ch)

**Parameters**  -> ch  A valid character.

**Result**  Returns true if the character is not a control character, false otherwise.

**Comments**  This macro differs from TxtCharIsGraph in that it returns true if the character is whitespace. TxtCharIsGraph returns false if the character is whitespace.

If you are using a debug ROM and you pass a virtual character to this macro, a fatal alert is generated.

**Compatibility**  Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueCharIsPrint. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**  TxtCharIsValid
**TxtCharIsPunct**

**Purpose**
Macro that indicates if a character is a punctuation mark.

**Declared In**
TxtMgr.h

**Prototype**
TxtCharIsPunct (ch)

**Parameters**
-> ch A valid character.

**Result**
Returns true if the character is a punctuation mark, false otherwise.

**Compatibility**
Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueCharIsPunct. For more information, see Chapter 75, “PalmOSGlue Library.”

---

**TxtCharIsSpace**

**Purpose**
Macro that indicates if a character is a whitespace character.

**Declared In**
TxtMgr.h

**Prototype**
TxtCharIsSpace (ch)

**Parameters**
-> ch A valid character.

**Result**
Returns true if the character is whitespace such as a blank space, tab, or newline; false otherwise.

**Compatibility**
Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueCharIsSpace. For more information, see Chapter 75, “PalmOSGlue Library.”
**TxtCharIsUpper**

**Purpose**     Macro that indicates if a character is an uppercase letter.

**Declared In** TextMgr.h

**Prototype**  TxtCharIsUpper (ch)

**Parameters** -> ch         A valid character.

**Result**     Returns true if the character is an uppercase letter, false otherwise.

**Compatibility** Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueCharIsUpper. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**   TxtCharIsAlpha, TxtCharIsLower

**TxtCharIsValid**

**Purpose** Determine whether a character is valid character given the Palm OS character encoding.

**Declared In** TextMgr.h

**Prototype** Boolean TxtCharIsValid (WChar inChar)

**Parameters** -> inChar     A character.

**Result**     Returns true if inChar is a valid character; false if inChar is not a valid character.

**Compatibility** Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS,
link with the PalmOSGlue library and call TxtGlueCharIsValid. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**  
TxtCharAttr, TxtCharIsPrint

## TxtCharSize

**Purpose**  
Return the number of bytes required to store the character in a string.

**Declared In**  
TextMgr.h

**Prototype**  
UInt16 TxtCharSize (WChar inChar)

**Parameters**  
\( \rightarrow \text{inChar} \)  
A valid character.

**Result**  
The number of bytes required to store the character in a string.

**Comments**  
Although character variables are always two-byte long WChar values, in some character encodings such as Shift-JIS, characters in strings are represented by a mix of one or more bytes per character. If the character can be represented by a single byte (its high-order byte is 0), it is stored in a string as a single-byte character.

**Compatibility**  
Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueCharSize. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**  
TxtCharBounds
## TxtCharWidth

**Purpose**
Return the width required to display the specified character in the current font. If the specified character does not exist within the current font, the missing character symbol is substituted.

**Declared In**
TextMgr.h

**Prototype**
```c
Int16 TxtCharWidth (WChar inChar)
```

**Parameters**
- `inChar`
  A valid character.

**Result**
Returns the width of the specified character (in pixels).

**Comments**
Use `FntWCharWidth` or `FntGlueWCharWidth` instead of this routine.

**Compatibility**
Implemented only if International Feature Set is present.

## TxtCharXAttr

**Purpose**
Return the extended attribute bits for a character.

**Declared In**
TextMgr.h

**Prototype**
```c
UInt16 TxtCharXAttr (WChar inChar)
```

**Parameters**
- `inChar`
  A valid character.

**Result**
Returns an unsigned 16-bit value with one or more extended attribute bits set. For specific return values, look in the header files that are specific to certain character encodings (`CharLatin.h` or `CharShiftJIS.h`).

**Comments**
To interpret the results, you must know the character encoding being used. Use `FtrGet` with `sysFtrNumEncoding` as the feature number to determine the character encoding. This returns one of the `CharEncodingType` values. For example:
WChar ch;
UInt16 attr;
UInt32 encoding;
...
attr = TxtCharXAttr(ch);
if (FtrGet(sysFtrCreator, sysFtrNumEncoding,
   &encoding) != errNone)
   encoding = charEncodingPalmLatin;
if (encoding == charEncodingUTF8) {
}

Compatibility
Implemented only if International Feature Set is present. To use this
function in code intended to be run on earlier versions of Palm OS,
link with the PalmOSGlue library and call TxtGlueCharXAttr.
For more information, see Chapter 75, "PalmOSGlue Library."

See Also  
TxtCharAttr

TxtCompare

Purpose
Performs a case-sensitive comparison of all or part of two text
buffers.

Declared In
TextMgr.h

Prototype
Int16 TxtCompare (const Char* s1, UInt16 s1Len,
   UInt16* s1MatchLen, const Char* s2, UInt16 s2Len,
   UInt16* s2MatchLen)

Parameters
-> s1  Pointer to the first text buffer to compare.
-> s1Len  The length in bytes of the text pointed to by s1.
<- s1MatchLen  Points to the offset of the first character in s1
   that determines the sort order. Pass NULL for
   this parameter if you don’t need to know this
   number.
-> s2  Pointer to the second text buffer to compare.
-> s2Len  The length in bytes of the text pointed to by s2.
<- s2MatchLen  Points to the offset of the first character in s2 that determines the sort order. Pass NULL for this parameter if you don’t need to know this number.

**Result**  Returns one of the following values:

- `< 0`  If s1 occurs before s2 in alphabetical order.
- `> 0`  If s1 occurs after s2 in alphabetical order.
- `0`  If the two substrings that were compared are equal.

**Comments**  This function performs a case-sensitive comparison. If you want to perform a case-insensitive comparison, use `TxtCaselessCompare`.

The `s1MatchLen` and `s2MatchLen` parameters are not as useful for the `TxtCompare` function as they are for the `TxtCaselessCompare` function because `TxtCompare` implements a multi-pass sort algorithm. (See the Compatibility section below for further details.) For example, comparing the string “celery” with the string “Cauliflower” returns a positive value to indicate that “celery” sorts after “Cauliflower,” and it returns a match length of 1 to indicate that the second letter determines the sort order (“e” comes after “a”). However, because `TxtCompare` ultimately does a case-sensitive comparison, comparing the string “c” to the string “C” produces a negative result and a match length of 0.

In certain character encodings (such as Shift JIS), one character may be accurately represented as either a single-byte character or a multi-byte character. `TxtCompare` accurately matches a single-byte character with its multi-byte equivalent. For this reason, the values returned in `s1MatchLen` and `s2MatchLen` are not always equal.

You must make sure that the parameters s1 and s2 point to the start of a a valid character. That is, they must point to the first byte of a multi-byte character or they must point to a single-byte character; if they don’t, results are unpredictable.

**Compatibility**  Implemented only if **International Feature Set** is present. To use this function in code intended to be run on earlier versions of Palm OS,
link with the PalmOSGlue library and call TxtGlueCompare. For more information, see Chapter 75, “PalmOSGlue Library.”

Prior to Palm OS 4.0, TxtCompare and StrCompare only performed one level of comparison and returned as soon as they found two unequal characters. For example, if you compared the string “celery” with the string “Cauliflower,” both functions returned a value indicating that “celery” should appear before “Cauliflower” because they sorted “c” before “C.”

In Palm OS 4.0, StrCompare callsTxtCompare, andTxtCompare performs a comparison using up to six comparison tables for sorting with increasing precision. As a result, in Palm OS 4.0 and higher, TxtCompare sorts “Cauliflower” before “celery.” The TxtGlueCompare function uses a two-pass sort on pre-4.0 devices, which will also sort “Cauliflower” before “celery.”

Palm OS 4.0 sorting of Shift-JIS characters attempts to duplicate the sorting algorithm described by the JIS standard.

In Palm OS 4.0, theTxtCompare function terminates when it finds a null byte in the string. In earlier releases, it terminated only when it reached the ending byte specified by the length parameters.

See Also StrCompare, TxtFindString
TxtConvertEncoding

**Purpose**
Convert a text buffer from one character encoding to another.

**Declared In**
TextMgr.h

**Prototype**
Err TxtConvertEncoding (Boolean newConversion,
TxtConvertStateType *ioStateP,
const Char *srcTextP, UInt16 *ioSrcBytes,
CharEncodingType srcEncoding, Char *dstTextP,
UInt16 *ioDstBytes, CharEncodingType dstEncoding,
const Char *substitutionStr,
UInt16 substitutionLen)

**Parameters**

- **newConversion**
  true if this function call is starting a new conversion, or false if this function call is a continuation of a previous conversion.

- **ioStateP**
  If newConversion is false, this parameter must point to the same data used for the previous invocation. If newConversion is true and no subsequent calls are planned, this parameter can be NULL.

- **srcTextP**
  A pointer to a text buffer. If newConversion is true, this must point to the start of a text buffer. If newConversion is false, it may point to a location in the middle of a text buffer. In either case, it must point to an inter-character boundary.

- **ioSrcBytes**
  A pointer to the length in bytes of the text in srcTextP that needs to be converted. Upon return, contains the number of bytes successfully processed.

- **srcEncoding**
  The character encoding that srcTextP currently uses. This should be one of the Character Encoding Constants.
Text Manager
Text Manager Functions

<- dstTextP  A pointer to the destination text buffer or NULL. This should always point to the location where TxtConvertEncoding can begin writing.

<-> ioDstBytes A pointer to the length in bytes of dstTextP. Upon return, contains the number of bytes required to represent srcTextP in the new encoding.

-> dstEncoding The character encoding to which to convert srcTextP. This should be one of the Character Encoding Constants.

-> substitutionStr A string that should be used to substitute any invalid or inconvertible characters that occur in srcTextP. This string must already be in the destination encoding. If NULL, the function immediately returns when an invalid character is encountered.

-> substitutionLen The number of bytes in substitutionStr, not including the terminating null byte.

Result  Returns errNone upon success or one of the following if an error occurs:

txtErrConvertOverflow  The destination buffer is not large enough to contain the converted text.

txtErrConvertUnderflow  The end of the source buffer contains a partial character.

txtErrNoCharMapping  The device does not contain a mapping between the source and destination encodings for at least one of the characters in srcTextP.
txtErrUnknownEncoding

One of the specified encodings is not valid. Currently, both the source and destination encodings must match either the device’s encoding or one of the Unicode character encodings.

Comments

This function converts ioSrcBytes of text in srcTextP from the srcEncoding to the dstEncoding character encoding and returns the result in dstTextP. Currently, the focus of TxtConvertEncoding is to convert between Unicode-encoded text and the device’s character encoding. For this reason, TxtConvertEncoding can only handle conversions between the device’s encoding and one of UTF-8, UCS-2, UTF-16LE, or UTF-16BE. If you specify any other character encoding for either the source or the destination buffer, the error code txtErrUnknownEncoding is returned.

You can retrieve the device’s encoding using the following function:

FtrGet(sysFtrCreator, sysFtrNumEncoding, &encoding)

If you’re converting text that was received from the Internet, the encoding’s name is passed along with the text data. Use the TxtNameToEncoding function to convert the name to a CharEncodingType value.

The dstTextP buffer must be large enough to hold the result of converting srcTextP to the specified encoding. You can pass NULL for the dstTextP parameter to determine the required length of the buffer before actually doing the conversion. (The required length is returned in ioDstBytes.)

If the function encounters an inconvertible character in the source text, it puts substitutionStr in the destination buffer in that character’s place and continues the conversion. When the conversion is complete, it returns txtErrNoCharMapping to indicate that an error occurred. If substitutionStr is NULL, the function stops the conversion and immediately returns txtErrNoCharMapping. ioSrcBytes is set to the offset of the inconvertible character, dstTextP contains the converted string up
to that point, and ioDstBytes contains the size of the converted text. You can examine the character at ioSrcBytes and choose to move past it and continue the conversion. Follow the rules for making repeated calls to TxtConvertEncoding as described in the next paragraph.

You can make repeated calls to TxtConvertEncoding in a loop if you only want to convert part of the input buffer at a time. When you make repeated calls to this function, the first call should use true for newConversion, and srcTextP should point to the start of the text buffer. All subsequent calls should use the following values:

- **newConversion** false.
- **ioStateP** The same data that was returned by the previous invocation.
- **srcTextP** The location where this call should begin converting. Typically, this would be the previous srcTextP plus the number of bytes returned in ioSrcBytes.
  
  If you are skipping over an inconvertible character, srcTextP must point to the character after that location.

  <-> **ioSrcBytes** The number of bytes that this function call should convert.

- **dstTextP** A pointer to a location where this function can begin writing the converted string. You might choose to have each function call write to a different destination buffer. To have successive calls write to the same buffer, pass the previous dstTextP plus the number of bytes returned in ioDstBytes each time.

- **ioDstBytes** The number of bytes available for output in the dstTextP buffer. In other words, the number of bytes remaining.

**Compatibility** Implemented only if 4.0 New Feature Set is present.
TxtEncodingName

**Purpose**
Obtain a character encoding’s name.

**Declared In**
TextMgr.h

**Prototype**
```c
const Char* TxtEncodingName(CharEncodingType inEncoding)
```

**Parameters**
- `inEncoding`
  One of the values from `CharEncodingType`, indicating a character encoding.

**Result**
A constant string containing the name of the encoding. The possible return values are defined in PalmLocale.h. They are:

- `encodingNameAscii`   us ascii
- `encodingNameISO8859_1` ISO-8859-1
- `encodingNameCP1252`   ISO-8859-1-Windows-3.1-Latin-1
- `encodingNameShiftJIS` Shift_JIS
- `encodingNameCP932`    Windows-31J
- `encodingNameUTF8`     UTF-8
- `""`                    The encoding is not known

**Comments**
Use this function to obtain the official name of the character encoding, suitable to pass to an Internet application or any other application that requires the character encoding’s name to be passed along with the data.

**Compatibility**
Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueEncodingName. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**
TxtNameToEncoding
TxtFindString

**Purpose**
Perform a case-insensitive search for a string in another string.

**Declared In**
TextMgr.h

**Prototype**
Boolean TxtFindString (const Char* inSourceStr, const Char* inTargetStr, UInt32* outPos, UInt16* outLength)

**Parameters**
- -> inSourceStr Pointer to the string to be searched.
- -> inTargetStr Prepared version of the string to be found. This string should either be passed directly from the strToFind field in the sysAppLaunchCmdFind launch code’s parameter block or it should be prepared using the PalmOSGlue function TxtGluePrepFindString.
- <- outPos Pointer to the offset of the match in inSourceStr.
- <- outLength Pointer to the length in bytes of the matching text.

**Result**
Returns true if the function finds inTargetStr within inSourceStr; false otherwise.

If found, the values pointed to by the outPos and outLength parameters are set to the starting offset and the length of the matching text. If not found, the values pointed to by outPos and outLength are set to 0.

The search that TxtFindString performs is locale-dependent. On most ROMs with Latin-based encodings, TxtFindString returns true only if the string is at the beginning of a word. On Shift-JIS encoded ROMs, TxtFindString returns true if the string is located anywhere in the word.

**Comments**
Use this function instead of FindStrInStr to support the global system find facility. This function contains an extra parameter,
outLength, to specify the length of the text that matched. Pass this value to `FindSaveMatch` in the appCustom parameter. Then when your application receives `sysAppLaunchCmdGoTo`, the matchCustom field contains the length of the matching text. You use the length of matching text to highlight the match within the selected record.

You must make sure that the parameters `inSourceStr` and `inTargetStr` point to the start of a valid character. That is, they must point to the first byte of a multi-byte character, or they must point to a single-byte character; if they don’t, results are unpredictable.

**Compatibility**

Implemented only if [International Feature Set](https://developer.palm.com) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueFindString`. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**

[TxtCaselessCompare](#)

---

## TtxtGetChar

**Purpose**

Retrieve the character starting at the specified offset within a text buffer.

**Declared In**

`TextMgr.h`

**Prototype**

```c
WChar TtxtGetChar (const Char* inText, UInt32 inOffset)
```

**Parameters**

- `inText` Pointer to the text buffer to be searched.
- `inOffset` A valid offset into the buffer `inText`. This offset must point to an inter-character boundary.

**Result**

Returns the character at `inOffset` in `inText`.

**Comments**

You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-
byte character or it must point to a single-byte character; if it
doesn’t, results are unpredictable.

**Compatibility**
Implemented only if [International Feature Set](#) is present. To use this
function in code intended to be run on earlier versions of Palm OS,
link with the PalmOSGlue library and call `TxtGlueGetChar`. For
more information, see Chapter 75, "PalmOSGlue Library."

**See Also**
`TxtGetNextChar`, `TxtSetNextChar`

### **TxtGetNextChar**

**Purpose**
Retrieve the character starting at the specified offset within a text
buffer.

**Declared In**
`TextMgr.h`

**Prototype**
```c
UInt16 TxtGetNextChar (const Char* inText, 
UInt32 inOffset, WChar* outChar)
```

**Parameters**
- `- > inText` Pointer to the text buffer to be searched.
- `- > inOffset` A valid offset into the buffer `inText`. This
  offset must point to an inter-character boundary.
- `< - outChar` The character at `inOffset` in `inText`. Pass
  NULL for this parameter if you don’t need the
  character returned.

**Result**
Returns the size in bytes of the character at `inOffset`. If `outChar`
is not NULL upon entry, it points to the character at `inOffset` upon
return.

**Comments**
You can use this function to iterate through a text buffer character-
by-character in this way:
```c
UInt16 i = 0;
WChar ch;
while (i < bufferLength) {
```
i += TxtGetNextChar(buffer, i, &ch);
    //do something with ch.
}

You must make sure that the parameter inText points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn’t, results are unpredictable.

**Compatibility**
Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueGetNextChar. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**
TxtGetChar, TxtGetPreviousChar, TxtSetNextChar

---

**TxtGetPreviousChar**

**Purpose**
Retrieve the character before the specified offset within a text buffer.

**Declared In**
TextMgr.h

**Prototype**

```
UInt16 TxtGetPreviousChar (const Char* inText, Uint32 inOffset, WChar* outChar)
```

**Parameters**

- `inText`: Pointer to the text buffer to be searched.
- `inOffset`: A valid offset into the buffer inText. This offset must point to an inter-character boundary.
- `outChar`: The character immediately preceding inOffset in inText. Pass NULL for this parameter if you don’t need the character returned.

**Result**
Returns the size in bytes of the character preceding inOffset in inText. If outChar is not NULL upon entry, then it points to the character preceding inOffset upon return. Returns 0 if inOffset is at the start of the buffer (that is, inOffset is 0).
Comments

You can use this function to iterate through a text buffer character-by-character in this way:

```c
WChar ch;
/* Find the start of the character containing
the last byte. */
TxtCharBounds (buffer, bufferLength - 1,
&start, &end);
i = start;
while (i > 0) {
    i -= TxtGetPreviousChar(buffer, i, &ch);
    //do something with ch.
}
```

This function is often slower to use than `TxtGetNextChar` because it must determine the appropriate character boundaries if the byte immediately before the offset is valid in more than one location (start, middle, or end) of a multi-byte character. To do this, it must work backwards toward the beginning of the string until it finds an unambiguous byte.

You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn’t, results are unpredictable.

Compatibility

Implemented only if `International Feature Set` is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueGetPreviousChar`. For more information, see Chapter 75, “PalmOSGlue Library.”
**TxtGetTruncationOffset**

**Purpose**

Return the appropriate byte position for truncating a text buffer such that it is at most a specified number of bytes long.

**Declared In**

TextMgr.h

**Prototype**

```c
UInt32 TxtGetTruncationOffset (const Char* inText, UInt32 inOffset)
```

**Parameters**

- `inText` Pointer to a text buffer.
- `inOffset` An offset into the buffer `inText`.

**Result**

Returns the appropriate byte offset for truncating `inText` at a valid inter-character boundary. The return value may be less than or equal to `inOffset`.

**Comments**

You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn’t, results are unpredictable.

**Compatibility**

Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueGetTruncationOffset`. For more information, see Chapter 75, “PalmOSGlue Library.”

**TxtGetWordWrapOffset**

**Purpose**

Locate an appropriate place for a line break in a text buffer.

**Declared In**

TextMgr.h

**Prototype**

```c
UInt32 TxtGetWordWrapOffset (const Char *iTextP, UInt32 iOffset)
```

**Parameters**

- `iTextP` Pointer to a text buffer.
-> iOffset  

Pointer to the offset where the search should begin. The search is performed backward starting from this offset.

**Result**

Returns the offset of a character that can begin on a new line (typically, the beginning of the word that contains \texttt{iOffset} or last word before \texttt{iOffset}). If an appropriate break could not be found, returns \texttt{iOffset}.

**Comments**

The \texttt{FntWordWrap} and \texttt{FntWordWrapReverseNLines} functions call \texttt{TxtGetWordWrapOffset} to locate an appropriate place to break the text. The returned offset points to the character that should begin the next line.

This function starts at \texttt{iOffset} and works backward until it finds a character that typically occurs between words (for example, white space or punctuation). Then it moves forward until it locates the character that begins a word (typically, a letter or number). Note that this function may return an offset value that is greater than the one passed in if the offset passed in occurs immediately before white space or in the middle of white space.

**Compatibility**

Implemented only if \texttt{4.0 New Feature Set} is present.

**See Also**

\texttt{TxtWordBounds}

**TxtMaxEncoding**

**Purpose**

Return the higher of two encodings.

**Declared In**

\texttt{TextMgr.h}

**Prototype**

\texttt{CharEncodingType TxtMaxEncoding(CharEncodingType a, CharEncodingType b)}

**Parameters**

\texttt{-> a}  

A character encoding to compare.
-> b

Another character encoding to compare.

**Result**

Returns the higher of a or b. One character encoding is higher than another if it is more specific. For example code page 1252 is “higher” than ISO 8859-1 because it represents more characters than ISO 8859-1.

**Comments**

This function is used by `TxtStrEncoding` to determine the encoding required for a string.

**Compatibility**

Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueMaxEncoding`. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**

`TxtCharEncoding`, `CharEncodingType`  

**TxtNameToEncoding**

**Purpose**

Return an encoding’s constant given its name.

**Declared In**

`TextMgr.h`

**Prototype**

```
CharEncodingType TxtNameToEncoding( const Char *iEncodingName )
```

**Parameters**

-> iEncodingName

One of the string constants containing the official name of an encoding. See `TxtEncodingName` for a list.

**Result**

Returns one of the Character Encoding Constants. Returns `charEncodingUnknown` if the specified encoding could not be found.

**Comments**

Use this function to convert a character encoding’s name as received from an Internet application into the character encoding constant that some text manager functions require.
This function properly converts aliases for a character encoding. For example, passing the strings “us-ascii”, “ASCII”, “cp367”, and “IBM367” all return charEncodingAscii.

The known character encodings are device-dependent. For example, a device with the Shift-JIS encoding will not know all of the aliases for Latin character encodings; however, it will know all of the aliases for Shift-JIS.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.

**See Also**

TxtEncodingName

---

**TxtNextCharSize**

**Purpose**

Macro that returns the size of the character starting at the specified offset within a text buffer.

**Declared In**

TxtMgr.h

**Prototype**

TxtNextCharSize (inText, inOffset)

**Parameters**

- `-> inText` Pointer to the text buffer to be searched.
- `-> inOffset` A valid offset into the buffer inText. This offset must point to an inter-character boundary.

**Result**

Returns (as a UInt16) the size in bytes of the character at inOffset.

**Comments**

You must make sure that the parameter inText points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn’t, results are unpredictable.

**Compatibility**

Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with
the PalmOSGlue library and call TxtGlueNextCharSize. For more information, see Chapter 75, “PalmOSGlue Library.”

See Also TxtGetNextChar

**TxtParamString**

**Purpose** Replace substrings within a string with the specified values.

**Declared In** TextMgr.h

**Prototype**

```c
Char* TxtParamString (const Char* inTemplate, const Char* param0, const Char* param1, const Char* param2, const Char* param3)
```

**Parameters**

- `inTemplate` The string containing the substrings to replace.
- `param0` String to replace ^0 with or NULL.
- `param1` String to replace ^1 with or NULL.
- `param2` String to replace ^2 with or NULL.
- `param3` String to replace ^3 with or NULL.

**Result** Returns a pointer to a locked relocatable chunk in the dynamic heap that contains the appropriate substitutions.

**Comments**

This function searches `inTemplate` for occurrences of the sequences ^0, ^1, ^2, and ^3. When it finds these, it replaces them with the corresponding string passed to this function. Multiple instances of each sequence will be replaced.

The replacement strings can also contain the substitution strings, provided they refer to a later parameter. That is, the `param0` string contain have references to ^1, ^2, and ^3, the `param1` string can have references to ^2 and ^3, and the `param2` string can have references to ^3. Any other occurrences of the substitution strings in the replacement strings are ignored. For example, if `param3` is the string “^0”, any occurrences of ^3 in `inTemplate` are replaced with the string “^0”. 
**Text Manager**

**Text Manager Functions**

You must make sure that the parameter `inTemplate` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn’t, results are unpredictable.

`TxtParamString` allocates space for the returned string in the dynamic heap through a call to `MemHandleNew`, and then returns the result of calling `MemHandleLock` with this handle. Your code is responsible for freeing this memory when it is no longer needed.

**Compatibility**

Implemented if [3.5 New Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueParamString`. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**

`TxtReplaceStr`, `FrmCustomAlert`

---

**TxtPreviousCharSize**

**Purpose**

Macro that returns the size of the character before the specified offset within a text buffer.

**Declared In**

`TxtMgr.h`

**Prototype**

`TxtPreviousCharSize (inText, inOffset)`

**Parameters**

- `-> inText` Pointer to the text buffer.
- `-> inOffset` A valid offset into the buffer `inText`. This offset must point to an inter-character boundary.

**Result**

Returns (as a `UInt16`) the size in bytes of the character preceding `inOffset` in `inText`. Returns 0 if `inOffset` is at the start of the buffer (that is, `inOffset` is 0).

**Comments**

You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn’t, results are unpredictable.
This macro is often slower to use than `TxtNextCharSize` because it must determine the appropriate character boundaries if the byte immediately before the offset is valid in more than one location (start, middle, or end) of a multi-byte character. To do this, it must work backwards toward the beginning of the string until it finds an unambiguous byte.

**Compatibility**

Valid only if [International Feature Set](#) is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGluePreviousCharSize`. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**

`TxtGetPreviousChar`

### `TxtReplaceStr`

**Purpose**

Replace a substring of a given format with another string.

**Declared In**

`TextMgr.h`

**Prototype**

```c
UInt16 TxtReplaceStr (Char* ioStr, UInt16 inMaxLen, const Char* inParamStr, UInt16 inParamNum)
```

**Parameters**

- **ioStr**
  The string in which to perform the replacing.
- **inMaxLen**
  The maximum length in bytes that `ioStr` can become.
- **inParamStr**
  The string that `^inParamNum` should be replaced with. If NULL, no changes are made.
- **inParamNum**
  A single-digit number (0 to 9).

**Result**

Returns the number of occurrences found and replaced. Returns a fatal error message if `inParamNum` is greater than 9.

**Comments**

This function searches `ioStr` for occurrences of the string `^inParamNum`, where `inParamNum` is any digit from 0 to 9. When it finds the string, it replaces it with `inParamStr`. Multiple instances
will be replaced as long as the resulting string doesn’t contain more than inMaxLen bytes, not counting the terminating null.

You can set the inParamStr parameter to NULL to determine the required length of ioStr before actually doing the replacing. TxtReplaceStr returns the number of occurrences it finds of `^inParamNum. Multiply this value by the length of the inParamStr you intend to use to determine the appropriate length of ioStr.

You must make sure that the parameter ioStr points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn’t, results are unpredictable.

**Compatibility**

Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueReplaceStr. For more information, see Chapter 75, “PalmOSGlue Library.”

**TxtSetNextChar**

**Purpose**

Set a character within a text buffer.

**Declared In**

TextMgr.h

**Prototype**

UInt16 TxtSetNextChar (Char* ioText, UInt32 inOffset, WChar inChar)

**Parameters**

<- ioText Pointer to a text buffer.

-> inOffset A valid offset into the buffer inText. This offset must point to an inter-character boundary.

-> inChar The character to replace the character at inOffset with. Must not be a virtual character.

**Result**

Returns the size of inChar.
Comments  This function replaces the character in **ioText** at the location **inOffset** with the character **inChar**. Note that there must be enough space at **inOffset** to write the character.

You can use **TxtCharSize** to determine the size of **inChar**.

You must make sure that the parameter **ioText** points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn’t, results are unpredictable.

Compatibility  Implemented only if **International Feature Set** is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call **TxtGlueSetNextChar**. For more information, see Chapter 75, “PalmOSGlue Library.”

See Also  **TxtGetNextChar**

**TxtStrEncoding**

Purpose  Return the encoding required to represent a string.

Declared In  **TextMgr.h**

Prototype  ```
CharEncodingType TxtStrEncoding (const Char* inStr)
```

Parameters  

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inStr</td>
<td>A string.</td>
</tr>
</tbody>
</table>

Result  A **CharEncodingType** value that indicates the encoding required to represent **inChar**. If any character in the string isn’t recognizable, then **charEncodingUnknown** is returned.

Comments  The encoding for the string is the maximum encoding of any character in that string. For example, if a two-character string contains a blank space and a ü, the appropriate encoding is **charEncodingISO8859_1**. The blank space’s minimum encoding is ASCII. The minimum encoding for the ü is ISO 8859-1. The maximum of these two encodings is ISO 8859-1.
Because Palm OS only supports a single character encoding at a time, the results of this function is always logically equal to or less than the encoding used on the current system. That is, you’ll only receive a return value of charEncodingISO8859_1 if you’re running on a Latin-based system.

Use this function for informational purposes only. Your code should not assume that the character encoding returned by this function is the Palm OS system’s character encoding. (Instead use FtrGet as shown in the TxtCharXAttr function description.)

**Compatibility**

Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueStrEncoding. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**  
TxtCharEncoding, TxtMaxEncoding

**TxtTransliterate**

**Purpose**

Converts the specified number of bytes in a text buffer using the specified operation.

**Declared In**  
TextMgr.h

**Prototype**

Err TxtTransliterate (const Char* inSrcText, UInt16 inSrcLength, Char* outDstText, UInt16* ioDstLength, TranslitOpType inOp)

**Parameters**

- **-> inSrcText** Pointer to a text buffer.
- **-> inSrcLength** The length in bytes of inSrcText.
- **<- outDstText** The output buffer containing the converted characters.
- **<-iODstLength** Upon entry, the maximum length of outDstText. Upon return, the actual length of outDstText.
-> inOp A 16-bit unsigned value that specifies which transliteration operation is to be performed. The values possible for this field are specific to the character encoding used on a particular device. These operations are universally available:

translitOpUpperCase
Converting the character to uppercase letters.

translitOpLowerCase
Converting the characters to lowercase letters.

translitOpPreprocess
Don’t actually perform the operation. Instead, return in ioDstLength the amount of space required for the output text.

Result Returns one of the following values:

errNone Success

txtErrUnknownTranslitOp inOp’s value is not recognized

txtErrTranslitOverrun
inSrcText and outDstText point to the same memory location and the operation has caused the function to overwrite unprocessed data in the input buffer.

txtErrTranslitOverflow
outDstText is not large enough to contain the converted string.

txtErrTranslitUnderflow
The end of the source buffer contains a partial character.

Comments inSrcText and outDstText may point to the same location if you want to perform the operation in place. However, you should
be careful that the space required for outDstText is not larger than inSrcText so that you don’t generate a txtErrTranslitOverrun error.

For example, suppose on a Shift JIS encoded system, you want to convert a series of single-byte Japanese Katakana symbols to double-byte Katakana symbols. You cannot perform this operation in place because it replaces a single-byte character with a multi-byte character. When the first converted character is written to the buffer, it overwrites the second input character. Thus, a text overrun has occurred.

You can ensure that you have enough space for the output by OR-ing your chosen operation with translitOpPreprocess. For example, to convert a string to uppercase letters, do the following:

```c
outSize = buf2Len;
error = TxtTransliterate(buf1, buf1len, &buf2, &outSize,
translitOpUpperCase|translitOpPreprocess);
if (outSize > buf2len)
/* allocate more memory for buf2 */
    error = TxtTransliterate(buf1, buf1Len, &buf2, &outSize, translitOpUpperCase);
```

You must make sure that the parameter inSrcText points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn’t, results are unpredictable.

**Compatibility**

Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueTransliterate. For more information, see Chapter 75, “PalmOSGlue Library.”

In Palm OS 4.0, the TxtTransliterate function terminates when it finds a null byte in the source string. In earlier releases, it terminated only when it reached the ending byte specified by the length parameter.
TxtWordBounds

**Purpose**
Find the boundaries of a word of text that contains the character starting at the specified offset.

**Declared In**
TextMgr.h

**Prototype**
Boolean TxtWordBounds (const Char* inText, UInt32 inLength, UInt32 inOffset, UInt32* outStart, UInt32* outEnd)

**Parameters**
- **inText**
  Pointer to a text buffer.
- **inLength**
The length in bytes of the text pointed to by inText.
- **inOffset**
  A valid offset into the text buffer inText. This offset must point to the beginning of a character.
- **outStart**
The starting offset of the text word.
- **outEnd**
The ending offset of the text word.

**Result**
Returns true if a word is found. Returns false if the word doesn’t exist or is punctuation or whitespace.

**Comments**
Assuming the ASCII encoding, if the text buffer contains the string “Hi! How are you?” and you pass 5 as the offset, TxtWordBounds returns the start and end of the word containing the character at offset 5, which is the character “o”. Thus, outStart and outEnd would point to the start and end of the word “How”.

You must make sure that the parameter inText points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn’t, results are unpredictable.

**Compatibility**
Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call TxtGlueWordBounds. For more information, see Chapter 75, “PalmOSGlue Library.”
In Palm OS 4.0, the `TxtWordBounds` function terminates when it finds a null byte in the string. In earlier releases, it terminated only when it reached the ending byte specified by the length parameter.

See Also

- `TxtCharBounds`
- `TxtCharIsDelim`
- `TxtGetWordWrapOffset`
Text Services Manager

This chapter provides information about the Text Services Manager API as declared in TextServicesMgr.h. The Text Services Manager provides the caller with an API for interacting with various text services, including front-end processors (FEPs), which are sometimes known as input methods. This chapter discusses the following topics:

- Text Services Manager Data Structures
- Text Services Manager Functions

Text Services Manager Data Structures

TsmFepModeType

The TsmFepModeType type specifies the input modes used by the functions TsmGetFepMode and TsmSetFepMode.

```c
typedef UInt16 TsmFepModeType;
```

TsmFepModeType can be set to one of the defined modes listed in the following table.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tsmFepModeDefault</td>
<td>0</td>
<td>The default input mode for the FEP. For example, with the Japanese FEP, the default mode is Hiragana.</td>
</tr>
</tbody>
</table>
Text Services Manager

Text Services Manager Functions

TsmGetFepMode

**Purpose**
Return the current input mode for the active front-end processor (FEP).

**Declared In**
TextServicesMgr.h

**Prototype**

```c
TsmFepModeType TsmGetFepMode (void *nullParam)
```

**Parameters**

- `nullParam` An unused status pointer that must be set to NULL.

**Result**
If there is an active FEP, returns the current mode for the active FEP. If there is no active FEP, returns `tsmFepModeOff`.

**Comments**
The most common use for this function is to save the current FEP mode. You could then call `TsmSetFepMode` to set the current mode to “off” and again to restore the saved mode once the application has finished using a special text field.

**Compatibility**
Implemented only if 3.5 New Feature Set is present. In Palm OS 3.5, the `nullParam` parameter takes a non-NUL value, allowing the caller to maintain its own status record. In Palm OS 4.0, this

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tsmFepModeOff</td>
<td>1</td>
<td>Indicates that there is no active FEP input mode (the FEP is off).</td>
</tr>
<tr>
<td>tsmFepModeCustom</td>
<td>128</td>
<td>A custom FEP input mode. You can have more than one custom mode; the starting value is 128. Katagana is an example of a custom input mode for the Japanese FEP.</td>
</tr>
</tbody>
</table>

Constant Value Description
parameter is unused and must be set to NULL. Any other value generates a non-fatal alert.

See Also  
TsmSetFepMode

TsmSetFepMode

**Purpose**
Set the input mode for the active front-end processor (FEP) to be the mode defined by the parameter inNewMode.

**Declared In**
TextServicesMgr.h

**Prototype**
TsmFepModeType TsmSetFepMode (void *nullParam, TsmFepModeType inNewMode)

**Parameters**
- `nullParam` An unused status pointer that must be set to NULL.
- `inNewMode` The new FEP input mode.

**Result**
Returns the previous input mode. If there is no active FEP, returns tsmFepModeOff.

**Comments**
The most common use for this function is to set the FEP mode to “off” while the application is using a special text field, and then to restore the previous mode. See TsmGetFepMode for more information on saving and restoring the FEP mode.

One common reason for explicitly disabling the FEP in code is when a text field will only contain 7-bit ASCII (numeric fields automatically turn off the FEP). For example, if the application has a password field and the contents of that field will always be 7-bit ASCII, the application should turn off the FEP to help prevent the user from entering invalid characters into the field.

Another common case occurs when the application has a numeric field, but cannot just rely on the numeric field attribute. For example, if you want the user to be able to enter the minus (“-”) sign, you cannot use a numeric field because the field code prevents the user from entering this character: its not a digit or a period. In this case, you should make it a regular field and have the
application screen the characters. The application should disable the FEP when such a pseudo-numeric field is active.

**IMPORTANT:** A mode change is currently enqueued as a keyDown event so that the field and FEP can remain properly synchronized and so that the mode change doesn’t affect any pending keyDown events. The mode change does not happen until the enqueued keyDown event is passed to FrmHandleEvent; if you call TsmGetFepMode immediately after calling TsmSetFepMode, you won’t see a mode change.

There are also some current limitations with changing the mode: there must be an active form; and if there is an active field in the form, it must not be a numeric field.

**Compatibility**  
Implemented only if 3.5 New Feature Set is present. In Palm OS 3.5, the nullParam parameter takes a non-NULL value, allowing the caller to maintain its own status record. In Palm OS 4.0, this parameter is unused and must be set to NULL. Any other value generates a non-fatal alert.

**See Also**  
TsmGetFepMode, EvtEnqueueKey, FrmHandleEvent
Time Manager

This chapter provides reference material for the time manager.

- Time Manager Data Structures
- Time Manager Functions

The header file `DateTime.h` declares the API that this chapter describes. For more information on the time manager, see the section “Time” in the Palm OS Programmer’s Companion, vol. I.

Time Manager Data Structures

The time manager uses these structures to store information.

**DateFormatType**

The `DateFormatType` enum specifies the different display formats for date values.

```c
typedef enum {
    dfMDYWithSlashes,
    dfDMYWithSlashes,
    dfDMYWithDots,
    dfDMYWithDashes,
    dfYMDWithSlashes,
    dfYMDWithDots,
    dfYMDWithDashes,
    dfMDYLongWithComma,
    dfDMYLong,
    dfDMYLongWithDot,
    dfDMYLongNoDay,
    dfDMYLongWithComma,
    dfYMDLongWithDot,
    dfYMDLongWithSpace,
    dfMYMed,
    dfMYMedNoPost,
```
dfMDYWithDashes

Value Descriptions

dfMDYWithSlashes  The month, day, and year numbers separated by slashes. For example, 12/31/95.
This is considered a short format.

dfDMYWithSlashes  The day, month, and year numbers separated by slashes. For example, 31/12/95.
This is considered a short format.

dfDMYWithDots  The day, month, and year numbers separated by dots. For example, 31.12.95.
This is considered a short format.

dfDMYWithDashes  The day, month, and year numbers separated by dashes. For example, 31-12-95.
This is considered a short format.

dfYMDWithSlashes  The year, month, and day numbers separated by slashes. For example, 95/12/31.
This is considered a short format.

dfYMDWithDots  The year, month, and day numbers separated by dots. For example, 95.12.31.
This is considered a short format.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dfYMDWithDashes</td>
<td>The year, month, and day numbers separated by dashes. For example, 95-12-31.</td>
</tr>
<tr>
<td></td>
<td>This is considered a short format.</td>
</tr>
<tr>
<td>dfMDYLongWithComma</td>
<td>The month, day, and year in long format, with a comma. For example, Dec 31, 1995.</td>
</tr>
<tr>
<td></td>
<td>This is considered a long format.</td>
</tr>
<tr>
<td>dfDMYLong</td>
<td>The month, day, and year in long format. For example, 31 Dec 1995.</td>
</tr>
<tr>
<td></td>
<td>This is considered a long format.</td>
</tr>
<tr>
<td>dfDMYLongWithDot</td>
<td>The month, day, and year in long format, with a dot. For example, 31. Dec 1995.</td>
</tr>
<tr>
<td></td>
<td>This is considered a long format.</td>
</tr>
<tr>
<td>dfDMYLongNoDay</td>
<td>The month and year in long format. For example, Dec 1995.</td>
</tr>
<tr>
<td></td>
<td>This is considered a long format.</td>
</tr>
<tr>
<td>dfDMYLongWithComma</td>
<td>The day, month, and year in long format, with a comma. For example, 31 Dec, 1995.</td>
</tr>
<tr>
<td></td>
<td>This is considered a long format.</td>
</tr>
<tr>
<td>dfYMDLongWithDot</td>
<td>The year, month, and day in long format with dot separators. For example, 1995.12.31.</td>
</tr>
<tr>
<td></td>
<td>This is considered a long format.</td>
</tr>
</tbody>
</table>
**Compatibility**
The `dfMDYWithDashes` constant is defined in Palm OS® 4.0 and higher.

**DateTimeType**
The `DateTimeType` structure represents a date and time value.

```c
typedef struct{
    Int16    second;
    Int16    minute;
    Int16    hour;
    Int16    day;
    Int16    month;
    Int16    year;
    Int16    weekDay;
} DateTimeType
```

```c
typedef DateTimeType *DateTimePtr;
```
Field Descriptions

second  The number of seconds. This is a value between 0 and 59.
minute  The number of minutes. This is a value between 0 and 59.
hour    The number of hours. This is a value between 0 and 23.
day     The day number. This is a value between 1 and 31.
month   The month number. This is a value between 1 and 12.
year    The year number.

weekDay The day number. This represents the number of days since Sunday and is thus a value between 0 and 6.

**DateTime**

The `DateTime` structure represents a date value.

```c
typedef struct{
    UInt16 year   :7;
    UInt16 month  :4;
    UInt16 day    :5;
} DateType;

typedef DateType *DatePtr;
```

Field Descriptions

year  The number of years since 1904.

Note that this is the format used on Macintosh computers.

month The month number. This is a value between 1 and 12.
day  The day number. This is a value between 1 and 31.

**DaylightSavingsTypes**

The `DaylightSavingsTypes` enum specifies the different forms of daylight savings times that you can specify for date and time values.
Note that the table uses “DST” to represent daylight savings time.

```c
typedef enum {
    dsNone,
    dsUSA,
    dsAustralia,
    dsWesternEuropean,
    dsMiddleEuropean,
    dsEasternEuropean,
    dsGreatBritain,
    dsRumania,
    dsTurkey,
    dsAustraliaShifted
} DaylightSavingsTypes;
```

**Value Descriptions**

- **dsNone**: No DST (daylight savings time)
- **dsUSA**: U.S.A. DST
- **dsAustralia**: Australian DST
- **dsWesternEuropean**: Western European DST
- **dsMiddleEuropean**: Middle European DST
- **dsEasternEuropean**: Eastern European DST
- **dsGreatBritain**: Great Britain and Eire DST
- **dsRumania**: Rumanian DST
- **dsTurkey**: Turkish DST
- **dsAustraliaShifted**: Australian DST, with the 1986 shift

**Compatibility**

If **4.0 New Feature Set** is present, this data type is obsolete. In versions 4.0 and higher, Palm OS represents daylight savings time as an integer value that gives the number of minutes to add to the current time for daylight savings time.
DayOfMonthType

The DayOfMonth enum specifies the different day-of-the-week numeric values that are returned by the DayOfMonth function. These values are used to represent repeating appointments that occur on specific days of the month; for example, the first Friday or the third Tuesday of each month.

typedef enum {
    dom1stSun, dom1stMon, dom1stTue, dom1stWen, dom1stThu, dom1stFri, dom1stSat,
    dom2ndSun, dom2ndMon, dom2ndTue, dom2ndWen, dom2ndThu, dom2ndFri, dom2ndSat,
    dom3rdSun, dom3rdMon, dom3rdTue, dom3rdWen, dom3rdThu, dom3rdFri, dom3rdSat,
    dom4thSun, dom4thMon, dom4thTue, dom4thWen, dom4thThu, dom4thFri, dom4thSat,
    domLastSun, domLastMon, domLastTue, domLastWen, domLastThu, domLastFri, domLastSat
} DayOfWeekType;

Value Descriptions

dom1stSun  The first Sunday of the month.
dom1stMon  The first Monday of the month.
dom1stTue  The first Tuesday of the month.
dom1stWen  The first Wednesday of the month.
dom1stThu  The first Thursday of the month.
dom1stFri  The first Friday of the month.
dom1stSat  The first Saturday of the month.
dom2ndSun  The second Sunday of the month.
dom2ndMon  The second Monday of the month.
dom2ndTue  The second Tuesday of the month.
dom2ndWen  The second Wednesday of the month.
dom2ndThu  The second Thursday of the month.
dom2ndFri  The second Friday of the month.
dom2ndSat  The second Saturday of the month.
dom3rdSun  The third Sunday of the month.
dom3rdMon  The third Monday of the month.
dom3rdTue  The third Tuesday of the month.
dom3rdWen  The third Wednesday of the month.
dom3rdThu  The third Thursday of the month.
dom3rdFri  The third Friday of the month.
dom3rdSat  The third Saturday of the month.
dom4thSun  The fourth Sunday of the month.
dom4thMon  The fourth Monday of the month.
dom4thTue  The fourth Tuesday of the month.
dom4thWen  The fourth Wednesday of the month.
dom4thThu  The fourth Thursday of the month.
dom4thFri  The fourth Friday of the month.
dom4thSat  The fourth Saturday of the month.
domLastSun The last Sunday of the month.
domLastMon The last Monday of the month.
domLastTue The last Tuesday of the month.
domLastWen The last Wednesday of the month.
domLastThu The last Thursday of the month.
domLastFri The last Friday of the month.
domLastSat The last Saturday of the month.
Compatibility

On Palm OS versions earlier than 4.0, this type was named DayOfWeekType.

**TimeFormatType**

The `TimeFormatType` enum specifies the different display formats for time values.

```c
typedef enum {
    tfColon,
    tfColonAMPM,
    tfColon24h,
    tfDot,
    tfDotAMPM,
    tfDot24h,
    tfHoursAMPM,
    tfHours24h,
    tfComma24h,
} TimeFormatType;

typedef TimeFormatType *TimeFormatPtr;
```

**Value Descriptions**

- **tfColon**: The hour and minutes separated by a colon character. For example, **1:00**.
- **tfColonAMPM**: The hour and minutes separated by a colon and followed by an AM/PM indication. For example, **1:00 pm**.
- **tfColon24h**: The 24-hour time with the hour and minutes separated by a colon character. For example, **13:00**.
- **tfDot**: The hour and minutes separated by a dot character. For example, **1.00**.
- **tfDotAMPM**: The hour and minutes separated by a period and followed by an AM/PM indication. For example, **1.00 pm**.
Time Manager

Time Manager Constants

The **TimeType** structure represents a time value.

```c
typedef struct {
    Uint8   hours;
    Uint8   minutes;
} TimeType;
typedef TimeType *TimePtr;
```

**Field Descriptions**

- **hours**  The number of hours. This is a value between 0 and 23.
- **minutes**  The number of minutes. This is value between 0 and 59.

**Time Manager Constants**

The following table shows the constants that represent the maximum lengths of strings returned by the date and time formatting routines `DateToAscii`, `DateToDOWDMFormat`, and `TimeToAscii`.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tfDot24h</td>
<td>The 24-hour time with the hour and minutes separated by a dot character. For example, 13.00.</td>
</tr>
<tr>
<td>tfHoursAMPM</td>
<td>The hour value followed by an AM/PM indication. For example, 1 pm.</td>
</tr>
<tr>
<td>tfHours24h</td>
<td>The 24-hour value, followed by an AM/PM indication. For example, 13.</td>
</tr>
<tr>
<td>tfComma24h</td>
<td>The 24-hour time with the hour and minutes separated by a comma character. For example, 13,00.</td>
</tr>
</tbody>
</table>
Time Manager Functions

DateAdjust

Purpose
Return a new date +/- the days adjustment.

Declared In
DateTime.h

Prototype
void DateAdjust (DatePtr dateP, Int32 adjustment)

Parameters
<- dateP  A pointer to a DateType structure with the date to be adjusted.
-> adjustment The number of days by which to adjust the date.

Result
Returns nothing. Upon return, dateP contains the adjusted date.

Constant Value Description

dateStringLength 9 Maximum length of the string returned by DateToAscii for short date formats.

longDateStrLength 15 Maximum length of the string returned by DateToAscii for medium and long date formats.

timeStringLength 9 Maximum length of the string returned by TimeToAscii.

dowDateStringLength 19 Maximum length of the string returned by DateToDOWDMFormat for short date formats.

dowLongDateStrLength 25 Maximum length of the string returned by DateToDOWDMFormat for both medium and long date formats.
Time Manager
Time Manager Functions

Comments This function advances the date and manages month and year wrapping conditions.

See Also TimAdjust

DateDaysToDate

Purpose Converts a date specified as the number of days since January 1, 1904 to a DateType structure.

Declared In DateTime.h

Prototype void DateDaysToDate (UInt32 days, DatePtr date)

Parameters
- days The number of days since 1/1/1904.
- date A pointer to a DateType structure that is updated with the computed date values.

Result Returns nothing. Upon return, the date information is returned in the structure referenced by the date parameter.

See Also DateSecondsToDate, DateToDays

DateSecondsToDate

Purpose Converts a date specified as the number of seconds since January 1, 1904 to a DateType structure.

Declared In DateTime.h

Prototype void DateSecondsToDate (UInt32 seconds, DatePtr date)

Parameters
- seconds The number of seconds since 1/1/1904.
<date>

A pointer to a <code>DateType</code> structure that is updated with the computed date values.

Result

Returns nothing. The structure referenced by the <code>date</code> parameter is updated with the date information.

See Also  
<code>DateDaysToDate</code>, <code>DateToDays</code>

**DateTemplateToAscii**

Purpose

Convert the specified date values into a string that is formatted according to a formatting template specification.

Declared In  
DateTime.h

Prototype

```c
UInt16 DateTemplateToAscii (const Char *templateP, 
UInt8 months, UInt8 days, UInt16 years, 
Char *stringP, Int16 stringLen)
```

Parameters

-<templateP>

A pointer to the template string used to format the date.

See the Comments section below for details on how to specify date formatting in this template string.

-<months>

The month number, which must be a value between 1 and 12.

-<days>

The day number, which must be a value between 1 and 31.

-<years>

The four-digit year number. For example, 1995.

<- stringP

A pointer to a string that is updated with the result.

If <code>stringP</code> is NULL, this function does not write an output string; however, it does return the length required for the output string.
If stringP is not NULL, this function writes the formatted string to stringP, writing up to stringSize bytes into stringP.

-> stringLen The size of the stringP buffer.

Result

The length of the formatted string, without the terminating null byte.

The DateTemplateToAscii returns the required length of the formatted string even if the stringP parameter is NULL; this allows you to determine the buffer size at runtime.

Comments

This function is intended as a replacement for the DateToAscii and DateToDOWDMFormat functions.

This function uses the formatting template referenced by templateP to create a formatted string from the date values that you pass in.

You specify a series of formatting substrings in templateP. Each substring has the form:

\[^<valueType><formatModifier>\]

Each substring has three components:

- The ^ character begins a substring.
- The <valueType> component is a single-digit value that specifies the value type.
- The <formatModifier> component is a single-letter value that specifies how you want that value formatted.

The following is an example of a template specification with three substrings:

\[^0z \^2l \^4r\]

Table 52.1 shows the values you can specify for the <valueType> component. Note that the formatted result depends on the <modifier> value.
Table 52.1 Template value types for the DateTemplateToAscii function

<table>
<thead>
<tr>
<th>Value</th>
<th>Value type</th>
<th>Formatted examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Day number</td>
<td>1, 01, 23, 31</td>
</tr>
<tr>
<td>1</td>
<td>Day name</td>
<td>Tue, Tuesday</td>
</tr>
<tr>
<td>2</td>
<td>Month name</td>
<td>May, Aug, August</td>
</tr>
<tr>
<td>3</td>
<td>Month number</td>
<td>4, 04, 11</td>
</tr>
<tr>
<td>4</td>
<td>Year number</td>
<td>97, 1997</td>
</tr>
</tbody>
</table>

Table 52.2 shows the values you can specify for the <modifier> component of each template substring.

Table 52.2 Template modifier types for the DateTemplateToAscii function

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>Formats the value in short form</td>
</tr>
<tr>
<td>r</td>
<td>Formats the value in regular form</td>
</tr>
<tr>
<td>l</td>
<td>Formats the value in long form</td>
</tr>
<tr>
<td>z</td>
<td>Adds a leading zero to the formatted numeric value</td>
</tr>
</tbody>
</table>

Finally, Table 52.3 shows examples of each value type formatted with each modifier type.

Table 52.3 Examples of formatted values

<table>
<thead>
<tr>
<th>Value type</th>
<th>Raw value</th>
<th>s (Short format)</th>
<th>r (Regular format)</th>
<th>l (Long format)</th>
<th>z (Zero format)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (Day number)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>02</td>
</tr>
<tr>
<td>1 (Day name)</td>
<td>2</td>
<td>T</td>
<td>Tue</td>
<td>Tuesday</td>
<td>n/a</td>
</tr>
</tbody>
</table>
For example, calling `DateTemplateToAscii` as follows:

```c
DateTemplateToAscii("^0z ^2l ^4r", 2, 7, 2000, myStr, 20)
```

Produces the following formatted string:

```
07 February 2000
```

**Compatibility**

Implemented only if 3.5 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `DateGlueTemplateToAscii`. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also**

`DateToAscii`, `DateToDOWDMFormat`
**DateToAscii**

**Purpose**  Convert the passed date to a string using the format specified by the `dateFormat` parameter.

**Declared In**  DateTime.h

**Prototype**  
```c
void DateToAscii (UInt8 months, UInt8 days, UInt16 years, DateFormatType dateFormat, Char *pString)
```

**Parameters**
- `-> months`  The month number, which must be a value between 1 and 12.
- `-> days`  The day number, which must be a value between 1 and 31.
- `-> years`  The four-digit year number. For example, 1995.
- `-> dateFormat`  Any `DateFormatType` format.
- `<- pString`  A pointer to string that is updated with the result.

This string must be of length `dateStringLength` for short formats or `longDateStrLength` for medium or long formats. Note that these lengths do include the terminating null byte. For more information about required string lengths, see [TimeManager Constants](#).

**Result**  Returns nothing. The string reference by `pString` is updated with the formatted string.

**Comments**  If you are using a debug ROM, the string buffer is filled with either `dateStringLength` or `longStrLength` debugging bytes, depending on the value of the `dateFormat` parameter.

It is important to allocate enough space for your string buffer. Finding buffer overflow errors can be difficult when using a debug ROM. One common situation is when you pass a buffer that is too
small from a form, for an element such as a label or title. Then, the buffer overflow can clobber objects that follow the form in memory. When a form element’s location information is corrupted, it disappears from the display.

Note that you can use the DateTemplateToAscii function instead of this function if the 3.5 feature set is present. You can call the DateTemplateToAscii function with a NULL string buffer to predetermine the required size for your buffer.

See Also TimeToAscii, DateToDateDOWDMFormat, DateTemplateToAscii

**DateToDays**

**Purpose** Convert the DateType structure to the number of days elapsed from January 1, 1904.

**Declared In** DateTime.h

**Prototype**

```c
UInt32 DateToDays (DateType date)
```

**Parameters**

- `date` A DateType structure.

**Result** Returns the number of days elapsed from January 1, 1904 to the specified date.

**See Also** DateDaysToDate
**DateToDOWDMFormat**

**Purpose** Convert a date to a formatted string using the format specified by the `dateFormat` parameter. The resultant string includes the name of the day of the week.

**Declared In** DateTime.h

**Prototype**

```c
void DateToDOWDMFormat (UInt8 months, UInt8 days, UInt16 years, DateFormatType dateFormat, Char *pString)
```

**Parameters**

- `months` The month number, which must be a value between 1 and 12.
- `days` The day number, which must be a value between 1 and 31.
- `years` The four-digit year number. For example, 1995.
- `dateFormat` Any `DateFormatType` format.
- `pString` A pointer to a string that is updated with the result. The string must be of length `dowDateStringLength` for short formats or `dowLongDateStrLength` for medium or long date formats. See `Time Manager Constants` for string buffer lengths.

**Result** Returns nothing. The string referenced by `pString` is updated with the formatted string.

**Comments**

The values of some of the `Time Manager Constants` that specify the required string buffer lengths do change from time to time. You should always use the constants or verify the required lengths by checking the `datetime.h` file.

It is important to allocate enough space for your string buffer. Finding buffer overflow errors can be difficult when using a debug ROM. One common situation is when you pass a buffer that is too small from a form, for an element such as a label or title. Then, the
buffer overflow can clobber objects that follow the form in memory. When a form element's location information is corrupted, it disappears from the display.

Note that you can use the `DateTemplateToAscii` function instead of this function if the 3.5 feature set is present. You can call the `DateTemplateToAscii` function with a NULL string buffer to predetermine the required size for your buffer.

Compatibility  On Palm OS 3.1 Japanese ROMs, this function contains a bug that prevented it from properly displaying 4-byte long day names. To prevent this bug from affecting your application, use `DateGlueToDOWDMFormat` in the PalmOSGlue library instead of calling this function directly. For more information, see Chapter 75, “PalmOSGlue Library.”

See Also  `DateToAscii`, `DateTemplateToAscii`

### DayOfMonth

**Purpose**  Return a value that represents the day of a month on which the specified date occurs. The value represents a quantity such as “First Monday” or “Third Friday” as is used for repeating appointments in the Datebook.

**Declared In**  `DateTime.h`

**Prototype**  ```c
Int16 DayOfMonth (Int16 month, Int16 day, Int16 year)
```  

**Parameters**

- `- > month`  The month number, which must be a value between 1 and 12.

- `- > day`  The day number, which must be a value between 1 and 31.

- `- > year`  The four-digit year number. For example, 1995.

**Result**  Returns a value that represents day of the month. This value is one of the `DayOfMonthType` values.
The returns value can be used to specify on which day of the month an appointment repeats.

**DayOfWeek**

**Purpose**
Return the day of the week value for a specified date.

**Declared In**
DateTime.h

**Prototype**
Int16 DayOfWeek (Int16 month, Int16 day, Int16 year)

**Parameters**
- -> month The month number, which must be a value between 1 and 12.
- -> day The day number, which must be a value between 1 and 31.
- -> year The four-digit year number. For example, 1995.

**Result**
Returns one of the following values for the day of the week of the specified date, as shown in the following table:

<table>
<thead>
<tr>
<th>Day name</th>
<th>Returned day value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunday</td>
<td>0</td>
</tr>
<tr>
<td>Monday</td>
<td>1</td>
</tr>
<tr>
<td>Tuesday</td>
<td>2</td>
</tr>
<tr>
<td>Wednesday</td>
<td>3</td>
</tr>
<tr>
<td>Thursday</td>
<td>4</td>
</tr>
<tr>
<td>Friday</td>
<td>5</td>
</tr>
<tr>
<td>Saturday</td>
<td>6</td>
</tr>
</tbody>
</table>
**DaysInMonth**

**Purpose**  
Return the number of days in the month.

**Declared In**  
DateTime.h

**Prototype**  
Int16 DaysInMonth (Int16 month, Int16 year)

**Parameters**
- **-> month**  
The month number, which must be a value between 1 and 12.
- **-> year**  
The four-digit year number. For example, 1995.

**Result**  
Returns the number of days in the month for the specified year.

**TimAdjust**

**Purpose**  
Return a new date, with the time adjusted by the specified number of seconds.

**Declared In**  
DateTime.h

**Prototype**  
void TimAdjust (DateTimePtr dateTimeP, Int32 adjustment)

**Parameters**
- **<-> dateTimeP**  
A pointer to a **DataType** structure.
- **-> adjustment**  
The number of seconds by which to adjust the time.

**Result**  
Returns nothing. The structure referenced by dateTimeP is modified to contain the updated date and time.

**Comments**  
This function advances the time by the specified number of seconds and takes care of any wraparound conditions.

**See Also**  
DateAdjust
**TimDateTimeToSeconds**

**Purpose**
Return the number of seconds elapsed from 12:00 A.M. on January 1, 1904 to the specified date and time.

**Declared In**
DateTime.h

**Prototype**
UInt32 TimDateTimeToSeconds (DateTimePtr dateTimeP)

**Parameters**
- `-> dateTimeP` A pointer to a `DateTimeType` structure.

**Result**
The number of seconds elapsed from 12:00 A.M. on January 1, 1904 to the date referenced by `dateTimeP`.

**See Also**
TimSecondsToDateTime

**TimGetSeconds**

**Purpose**
Return the current date and time of the device in seconds since 12:00 A.M. on January 1, 1904.

**Declared In**
TimeMgr.h

**Prototype**
UInt32 TimGetSeconds (void)

**Parameters**
None.

**Result**
The number of seconds elapsed from 12:00 A.M. on January 1, 1904 to the current date and time on the device.

**See Also**
TimSetSeconds
**TimGetTicks**

**Purpose**
Return the tick count since the last reset. The tick count does not advance while the device is in sleep mode.

**Declared In**
TimeMgr.h

**Prototype**
UInt32 TimGetTicks (void)

**Parameters**
None.

**Result**
Returns the tick count.

**Comments**
You can call the `SysTicksPerSecond` routine to determine the number of ticks per second.

**See Also**
`SysTicksPerSecond`

---

**TimSecondsToDateTime**

**Purpose**
Converts a date specified as the number of seconds since January 1, 1904 to a `DateTimeType` structure.

**Declared In**
DateTime.h

**Prototype**
void TimSecondsToDateTime (UInt32 seconds, DateTimePtr dateTimeP)

**Parameters**
- `seconds` A date specified as the number of seconds elapsed from 12:00 A.M. on January 1, 1904 to the date
<- dateTimeP   A pointer to a DateTimeType structure that is updated with the date and time values.

**Result**

Returns nothing. The structure referenced by dateTimeP is updated with the date and time computed for the number of seconds since 12:00 A.M. on January 1, 1904.

**See Also**  TimDateTimeToSeconds

---

**TimSetSeconds**

**Purpose**

Set the clock of the device to the date and time passed as the number of seconds since 12:00 A.M. on January 1, 1904.

**Declared In**  TimeMgr.h

**Prototype**  void TimSetSeconds (UInt32 seconds)

**Parameters**

-> seconds   The number of seconds since 12:00 A.M. on January 1, 1904.

**Result**

Returns nothing.

**Comments**

If the Notification Feature Set is present, this function broadcasts the sysNotifyTimeChangeEvent to all interested parties. See Chapter 39, “Notification Manager,” for more information.

**See Also**  TimGetSeconds
**TimeToAscii**

**Purpose**  Convert the time to a string that is formatted according to the specified timeFormat.

**Declared In**  DateTime.h

**Prototype**  

```c
void TimeToAscii (UInt8 hours, UInt8 minutes, TimeFormatType timeFormat, Char *pString)
```

**Parameters**

- **hours**  The number of hours. This must be a value between 0 and 23.
- **minutes**  The number of minutes. This must be a value between 0 and 59.
- **timeFormat**  The time format for the resultant string. This must be one of the TimeFormatType values.
- **pString**  A pointer to a string that is updated with the resultant string. This string must be of length timeStringLength. See Time Manager Constants for information on string buffer lengths.

**Result**  Returns nothing. The string referenced by pString is updated with the formatted string.

**Comments**  If you are using a debug ROM in Palm OS 3.5, the string buffer is filled with timeStringLength debugging bytes.

It is important to allocate enough space for your string buffer. Finding buffer overflow errors can be difficult when using a debug ROM. One common situation is when you pass a buffer that is too small from a form, for an element such as a label or title. Then, the buffer overflow can clobber objects that follow the form in memory. When a form element’s location information is corrupted, it disappears from the display.

**See Also**  DateToAscii
TimeZoneToAscii

Purpose  Convert a time zone to a string.

Declared In  DateTime.h

Prototype  void TimeZoneToAscii (Int16 timeZone, const LmLocaleType *localeP, Char *string)

Parameters
- timeZone  A pointer to the time zone, given as minutes east of Greenwich Mean Time (GMT).
- localeP  A pointer to a locale (see LmLocaleType) that identifies the time zone country. You can use the constant LmAnyLanguage as the value for the language field of the structure pointed to by this parameter.
- string  A pointer to a string in which to return the result. This string must be of length timeZoneStringLength.

Result  Returns nothing.

Comments  This function returns a descriptive string for the specified time zone. This string identifies the time zone first by its country, such as “USA (Mountain)” or “Canada (Eastern).” If the function cannot find a time zone that matches the specified GMT offset and country, it returns a string containing the time zone as a numeric offset from the GMT (for example, “GMT+9:00”).

Compatibility  Implemented only if 4.0 New Feature Set is present.
TimTimeZoneToUTC

Purpose
Converts a date and time from a given time zone to Universal Coordinated Time (UTC). UTC is also known as Greenwich Mean Time (GMT).

Declared In
DateTime.h

Prototype
UInt32 TimTimeZoneToUTC (UInt32 seconds, Int16 timeZone, Int16 daylightSavingAdjustment)

Parameters
- `seconds` The number of seconds since 12:00 A.M. on January 1, 1904.
- `timeZone` The time zone, given as the number of minutes east of UTC. For time zones west of UTC but before the international dateline, this is a negative number.
- `daylightSavingAdjustment` The number of minutes to add to the current time for daylight savings time in this time zone.

Result
Returns the same time as `seconds` but in the Universal Coordinated Time. The value is still given as the number of seconds since 12:00 A.M. on January 1, 1904.

Comments
The returned value is not necessarily the time in Greenwich because Greenwich may be observing daylight saving time.

You can use this function to convert the local time to UTC. The time zone and the daylight savings adjustment are system preferences that can be retrieved using `PrefGetPreference`. For example, the following code converts the current local time to UTC:

```c
Int16 timeZone = PrefGetPreference(prefTimeZone);
Int16 daylightSavingAdjustment = PrefGetPreference(
    prefDaylightSavingAdjustment);
UInt32 utcTime =
```
TimeZoneToUTC(TimGetSeconds(), timeZone, daylightSavingAdjustment);

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also TimeZoneToUTC

TimUTCToTimeZone

Purpose Converts a date and time from Universal Coordinated Time (UTC) to the specified time zone. UTC is also known as Greenwich Mean Time (GMT).

Declared In DateTime.h

Prototype UInt32 TimUTCToTimeZone (UInt32 seconds, Int16 timeZone, Int16 daylightSavingAdjustment)

Parameters -> seconds The number of seconds since 12:00 A.M. on January 1, 1904 in UTC.

-> timeZone The time zone, given as the number of minutes east of UTC. For time zones west of UTC before the international dateline, this is a negative number.

-> daylightSavingAdjustment The number of minutes to add to the current time for daylight savings time in this time zone.

Result Returns the same time as seconds but in the specified time zone. The value is still given as the number of seconds since 12:00 A.M. on January 1, 1904.

Comments The seconds value is not necessarily the time in Greenwich because Greenwich may be observing daylight saving time.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also TimeZoneToUTC
Virtual File System Manager

The Virtual File System (VFS) Manager is a layer of software that manages all installed file system libraries. It provides a unified API to application developers while allowing them to seamlessly access many different types of file systems—such as VFAT, HFS, and NFS—on many different types of media, including Compact Flash, Memory Stick, and SmartMedia. This chapter provides reference material for the VFS Manager API as follows:

- VFS Manager Data Structures
- VFS Manager Constants
- VFS Manager Functions
- Application-Defined Functions

The header file `VFSMgr.h` declares the VFS Manager API. For more information on the VFS Manager, see Chapter 7, “Expansion,” in *Palm OS Programmer’s Companion*, vol. I.

Note that the VFS Manager is an optional system extension; the functions described in this chapter are implemented only if the VFS Manager Feature Set is present.

VFS Manager Data Structures

FileInfoType

The FileInfoType structure contains information about a specified file or directory. This information is returned as a parameter to `VFSDirEntryEnumerate`. The structure is defined as follows:

```c
typedef struct FileInfoTag {
    UInt32 attributes;
} FileInfoType;
```
Char *nameP;
UInt16 nameBufLen;
} FileInfoType, *FileInfoPtr;

Field Descriptions

attributes  Characteristics of the file or directory. See File and Directory Attributes for the bits that make up this field.

nameP  Pointer to the buffer that receives the full name of the file or directory. Initialize this parameter to NULL if you don’t want to receive the name.

nameBufLen  Size of the nameP buffer, in bytes.

FileRef

The FileRef type is used to encode references to files and directories.

typedef UInt32 FileRef;

VFSAnyMountParamType

The VFSAnyMountParamType structure is a base structure for VFSSlotMountParamType, VFSPOSEMountParamType, and other similar structures that may be defined in the future. Use one or the other according to how you set the mountClass parameter.

typedef struct VFSAnyMountParamTag {
    UInt16 volRefNum;
    UInt16 reserved;
    UInt32 mountClass;
} VFSAnyMountParamType;

typedef VFSAnyMountParamType *VFSAnyMountParamPtr;
Field Descriptions

volRefNum The volume reference number. This is initially obtained when you successfully mount a volume. It can then be used to format a volume with VFSVolumeFormat or unmount a volume with VFSVolumeUnmount.

reserved Reserved for future use.

mountClass Defines the type of mount to use with the specified volume. See Volume Mount Classes for a list of mount types.

VFSSlotMountParamType

The VFSSlotMountParamType structure is used when you are mounting a card located in an Expansion Manager slot. The vfsMountParam->mountClass field must be set to VFSMountClass_SlotDriver.

typedef struct VFSSlotMountParamTag {
    VFSAnyMountParamType vfsMountParam;
    UInt16 slotLibRefNum;
    UInt16 slotRefNum;
} VFSSlotMountParamType;

Field Descriptions

vfsMountParam See the description of VFSAnyMountParamType for an explanation of the fields in this structure. Set vfsMountParam->mountClass to VFSMountClass_SlotDriver to mount an Expansion Manager slot.

slotLibRefNum Reference number for the slot driver library allocated to the given slot number. Obtain this field by calling ExpSlotLibFind.

slotRefNum Number of the slot to be mounted.
**VFSPOSEMountParamType**

The VFSPOSEMountParamType structure is used when you are mounting a volume through the Palm OS® Emulator. The `vfsMountParam->mountClass` must be set to `VFSMountClass_POSE`. Note that ordinary applications and file systems shouldn’t use VFSPOSEMountParamType.

```c
typedef struct VFSPOSEMountParamTag {
    VFSAnyMountParamType vfsMountParam;
    UInt8 poseSlotNum;
} VFSPOSEMountParamType
```

**Field Descriptions**

- **vfsMountParam**
  - See the description of [VFSAnyMountParamType](#) for an explanation of the fields in this structure. Set `vfsMountParam->mountClass` to `VFSMountClass_POSE` to mount a virtual slot.

- **poseSlotNum**
  - Number of the virtual slot number to be mounted by the Palm OS Emulator.

**VolumeInfoType**

The VolumeInfoType structure defines information that is returned to [VFSVolumeInfo](#) and used throughout the VFS functions.

```c
typedef struct VolumeInfoTag {
    UInt32 attributes;
    UInt32 fsType;
    UInt32 fsCreator;
    UInt32 mountClass;
    UInt16 slotLibRefNum;
    UInt16 slotRefNum;
    UInt32 mediaType;
    UInt32 reserved;
} VolumeInfoType, *VolumeInfoPtr;
```
Field Descriptions

attributes
Characteristics of the volume. See Volume Attributes for the bits that make up this field.

fsType
File system type for this volume. See Defined File Systems for a list of the supported file systems.

fsCreator
Creator ID of this volume’s file system driver. This information is used with VFSCustomControl.

mountClass
Mount class that mounted this volume. The supported mount classes are listed under Volume Mount Classes.

slotLibRefNum
Reference to the slot driver library with which the volume is mounted. This field is only valid when the mount class is vfsMountClass_SlotDriver.

slotRefNum
Slot number where the card containing the volume is loaded. This field is only valid when the mount class is vfsMountClass_SlotDriver.

mediaType
Type of card media. See Defined Media Types in the Expansion Manager chapter for the list of values. This field is only valid when the mount class is vfsMountClass_SlotDriver.

reserved
Reserved for future use.

VFS Manager Constants

Defined File Systems
The following file systems are currently defined by the VFS Manager. These values are used with VFSVolumeInfo in the VolumeInfoType.fsType parameter.
### VFS Manager Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vfsFilesystemType_AFS</td>
<td>'afsu'</td>
<td>Unix Andrew file system</td>
</tr>
<tr>
<td>vfsFilesystemType_EXT2</td>
<td>'ext2'</td>
<td>Linux file system</td>
</tr>
<tr>
<td>vfsFilesystemType_FAT</td>
<td>'fats'</td>
<td>FAT12 and FAT16, which only handles 8.3 filenames</td>
</tr>
<tr>
<td>vfsFilesystemType_FFS</td>
<td>'ffsb'</td>
<td>Unix Berkeley block based file system</td>
</tr>
<tr>
<td>vfsFilesystemType_HFS</td>
<td>'hfss'</td>
<td>Macintosh standard hierarchical file system</td>
</tr>
<tr>
<td>vfsFilesystemType_HFSPlus</td>
<td>'hfse'</td>
<td>Macintosh extended hierarchical file system</td>
</tr>
<tr>
<td>vfsFilesystemType_HPFS</td>
<td>'hpfs'</td>
<td>OS2 High Performance file system</td>
</tr>
<tr>
<td>vfsFilesystemType_MFS</td>
<td>'mfso'</td>
<td>Macintosh original file system</td>
</tr>
<tr>
<td>vfsFilesystemType_NFS</td>
<td>'nfsu'</td>
<td>Unix Networked file system</td>
</tr>
<tr>
<td>vfsFilesystemType_Novell</td>
<td>'novl'</td>
<td>Novell file system</td>
</tr>
<tr>
<td>vfsFilesystemType_NTFS</td>
<td>'ntfs'</td>
<td>Windows NT file system</td>
</tr>
<tr>
<td>vfsFilesystemType_VFAT</td>
<td>'vfat'</td>
<td>FAT12 and FAT16 extended to handle long filenames</td>
</tr>
</tbody>
</table>
Open Mode Constants

This section describes constants that are used for the openMode parameter to the VFSFileOpen function. These constants specify the mode in which a file or directory is opened.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vfsModeExclusive</td>
<td>0x0001U</td>
<td>Open and lock the file or directory. This mode excludes anyone else from using the file or directory until it is closed.</td>
</tr>
<tr>
<td>vfsModeRead</td>
<td>0x0002U</td>
<td>Open for read access.</td>
</tr>
<tr>
<td>vfsModeWrite</td>
<td>0x0004U</td>
<td>vfsModeExclusive</td>
</tr>
<tr>
<td>vfsModeReadWrite</td>
<td>vfsModeWrite</td>
<td>vfsModeRead</td>
</tr>
<tr>
<td>vfsModeCreate</td>
<td>0x0008U</td>
<td>Create the file if it doesn't already exist. This open mode is implemented in the VFS layer, rather than in the file system library.</td>
</tr>
<tr>
<td>vfsModeTruncate</td>
<td>0x0010U</td>
<td>Truncate the file to zero (0) bytes after opening, removing all existing data. This open mode is implemented in the VFS layer, rather than in the file system library.</td>
</tr>
<tr>
<td>vfsModeVFSLayerOnly</td>
<td>vfsModeCreate</td>
<td>vfsModeTruncate</td>
</tr>
<tr>
<td>vfsModeLeaveOpen</td>
<td>0x0020U</td>
<td>Leave the file open even after the application exits.</td>
</tr>
</tbody>
</table>
Virtual File System Manager
VFS Manager Constants

File and Directory Attributes
The constants in the following table define bits that can be used individually or in combination when setting or interpreting the file attributes for a given file or directory. See VFSFileGetAttributes, VFSFileSetAttributes, and the FileInfoType data structure for specific use.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vfsFileAttrReadOnly</td>
<td>0x00000001UL</td>
<td>Read-only file or directory</td>
</tr>
<tr>
<td>vfsFileAttrHidden</td>
<td>0x00000002UL</td>
<td>Hidden file or directory</td>
</tr>
<tr>
<td>vfsFileAttrSystem</td>
<td>0x00000004UL</td>
<td>System file or directory</td>
</tr>
<tr>
<td>vfsFileAttrVolumeLabel</td>
<td>0x00000008UL</td>
<td>Volume label</td>
</tr>
<tr>
<td>vfsFileAttrDirectory</td>
<td>0x00000010UL</td>
<td>Directory</td>
</tr>
<tr>
<td>vfsFileAttrArchive</td>
<td>0x00000020UL</td>
<td>Archived file or directory</td>
</tr>
<tr>
<td>vfsFileAttrLink</td>
<td>0x00000040UL</td>
<td>Link to another file or directory</td>
</tr>
</tbody>
</table>

Volume Attributes
The constants in the following table define bits that can be used individually or in combination to make up the attributes field in the VolumeInfoType structure.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vfsVolumeAttrHidden</td>
<td>0x00000004UL</td>
<td>The volume should not be visible to the user.</td>
</tr>
<tr>
<td>vfsVolumeAttrReadOnly</td>
<td>0x00000002UL</td>
<td>The volume is read only.</td>
</tr>
<tr>
<td>vfsVolumeAttrSlotBased</td>
<td>0x00000001UL</td>
<td>Reserved. Check the mount class to determine how a volume is mounted.</td>
</tr>
</tbody>
</table>
Volume Mount Classes

The following constants define how a given volume is mounted. The mountClass field in the `VFSAnyMountParamType` and `VolumeInfoType` structures takes on one of these values.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vfsMountClass_POSE</td>
<td>'pose'</td>
<td>Mount the volume through the Palm OS Emulator. This is used for testing.</td>
</tr>
<tr>
<td>vfsMountClass_Simulator</td>
<td>sysFileTSimulator</td>
<td>Mount the volume through the simulator. This is used for testing.</td>
</tr>
<tr>
<td>vfsMountClass_SlotDriver</td>
<td>sysFileTSlotDriver</td>
<td>Mount the volume with a slot driver shared library.</td>
</tr>
</tbody>
</table>

Error Codes

The VFS Manager defines the following error codes:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vfsErrBadData</td>
<td>The operation could not be completed because of invalid data.</td>
</tr>
<tr>
<td>vfsErrBadName</td>
<td>Invalid filename, path, or volume label.</td>
</tr>
<tr>
<td>vfsErrBufferOverflow</td>
<td>The supplied buffer is too small.</td>
</tr>
<tr>
<td>vfsErrDirectoryNotFound</td>
<td>Returned when the path leading up to the file does not exist.</td>
</tr>
<tr>
<td>vfsErrDirNotEmpty</td>
<td>The directory is not empty and therefore cannot be deleted.</td>
</tr>
<tr>
<td>Constant</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>vfsErrFileAlreadyExists</td>
<td>A file with this name exists already in this location.</td>
</tr>
<tr>
<td>vfsErrFileBadRef</td>
<td>The file reference is invalid: it has been closed or was not obtained from VFSFileOpen.</td>
</tr>
<tr>
<td>vfsErrFileEOF</td>
<td>The file pointer is at the end of the file.</td>
</tr>
<tr>
<td>vfsErrFileGeneric</td>
<td>Generic file error.</td>
</tr>
<tr>
<td>vfsErrFileNotFound</td>
<td>The file was not found at the specified location.</td>
</tr>
<tr>
<td>vfsErrFilePermissionDenied</td>
<td>The requested permissions could not be granted.</td>
</tr>
<tr>
<td>vfsErrFileStillOpen</td>
<td>Returned from the underlying file system’s delete function if the file is still open.</td>
</tr>
<tr>
<td>vfsErrIsADirectory</td>
<td>This operation can only be performed on a regular file, not a directory.</td>
</tr>
<tr>
<td>vfsErrNameShortened</td>
<td>A volume name or filename was automatically shortened to conform to the file system specification.</td>
</tr>
<tr>
<td>vfsErrNoFileSystem</td>
<td>None of the installed file systems support this operation.</td>
</tr>
<tr>
<td>vfsErrNotADirectory</td>
<td>This operation can only be performed on a directory.</td>
</tr>
<tr>
<td>vfsErrVolumeBadRef</td>
<td>The volume reference number is invalid.</td>
</tr>
</tbody>
</table>
### VFSCustomControl

**Purpose**
Make a custom API call to a particular file system, given its creator ID. You can use `VFSVolumeInfo` to determine the creator ID of the file system for a given volume.

**Declared In**
VfsMgr.h

**Prototype**
```
Err VFSCustomControl(UInt32 fsCreator, 
                    UInt32 apiCreator, UInt16 apiSelector, 
                    void *valueP, UInt16 *valueLenP)
```

**Parameters**
- `-> fsCreator` Creator of the file system to call. A value of zero (0) tells the VFS Manager to check each registered file system, looking for one which supports the call.
- `-> apiCreator` Registered creator ID.
- `-> apiSelector` Custom operation to perform.
- `<-> valueP` A pointer to a buffer containing data specific to the operation. On exit, depending on the function of the particular custom call and on the value of `valueLenP`, the contents of this buffer may have been updated.

### Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vfsErrVolumeFull</td>
<td>There is insufficient space left on the volume.</td>
</tr>
<tr>
<td>vfsErrVolumeStillMounted</td>
<td>Returned from the underlying file system’s format function if the volume is still mounted.</td>
</tr>
</tbody>
</table>
VIRTUAL FILE SYSTEM MANAGER

VFS Manager Functions

valueLenP: On entry, points to the size of the valueP buffer. On exit, this value reflects the size of the data written to the valueP buffer. If valueLenP is NULL, valueP is passed to the file system but is not updated on exit.

Result
Returns one of the following error codes:

- **errNone**: No error.
- **expErrNotOpen**: The file system library necessary for this call has not been installed or has not been opened.
- **expErrUnsupportedOperation**: The specified opcode and/or creator is unsupported or undefined.
- **sysErrParamErr**: The valueP buffer is too small.
- **vfsErrNoFileSystem**: VFS Manager cannot find an appropriate file system to handle the request.

Comments
The driver identifies the call and its API by a registered creator ID and a selector. This allows file system developers to extend the API by defining selectors for their creator IDs. It also allows file system developers to support selectors (and custom calls) defined by other file system developers.

This function must return **expErrUnsupportedOperation** for all unsupported or undefined opcodes and/or creators.

Compatibility
Implemented only if the [VFS Manager Feature Set](#) is present.
VFSDirCreate

**Purpose**  
Create a new directory.

**Declared In**  
VfsMgr.h

**Prototype**  
Err VFSDirCreate(UInt16 volRefNum,  
const Char *dirNameP)

**Parameters**
- `-> volRefNum`  
Volume reference number returned from VFSVolumeEnumerate.
- `-> dirNameP`  
Pointer to the full path of the directory to be created.

**Result**  
Returns one of the following error codes:
- `errNone`  
No error
- `expErrNotOpen`  
The file system library necessary for this call has not been installed or has not been opened.
- `vfsErrBadName`  
Some or all of the path, up to but not including the last component specified in the `dirNameP` parameter, does not exist.
- `vfsErrFileAlreadyExists`  
A file with this name already exists in this location.
- `vfsErrNoFileSystem`  
The VFS Manager cannot find an appropriate file system to handle the request.
- `vfsErrVolumeBadRef`  
The volume has not been mounted.
- `vfsErrVolumeFull`  
There is not enough space left on the volume.

**Comments**  
All parts of the path except the last component must already exist. The `vfsFileAttrDirectory` attribute is set with this function.
**Virtual File System Manager**

**VFS Manager Functions**

VFSDirCreate does not open the directory. Any operations you want to perform on this directory require a reference, which is obtained through a call to VFSFileOpen.

**Compatibility**

Implemented only if the [VFS Manager Feature Set](#) is present.

**VFSDirEntryEnumerate**

**Purpose**

Enumerate the entries in a given directory. Entries can include files, links, and other directories.

**Declared In**

VfsMgr.h

**Prototype**

Err VFSDirEntryEnumerate (FileRef dirRef, UInt32 *dirEntryIteratorP, FileInfoType *infoP)

**Parameters**

- `-> dirRef` Directory reference returned from VFSFileOpen.
- `<-> dirEntryIteratorP` Pointer to the index of the last entry enumerated. For the first iteration, initialize this parameter to the constant vfsIteratorStart. Upon return, this references the next entry in the directory. If infoP is the last entry, this parameter is set to vfsIteratorStop.
- `<-> infoP` Pointer to the FileInfoType data structure that contains information about the given directory entry. The nameP and nameBufLen fields in this structure must be initialized prior to calling VFSDirEntryEnumerate.

**Result**

Returns one of the following error codes:

- `errNone` No error.
- `expErrEnumerationEmpty` There are no directory entries left to enumerate.
expErrNotOpen  The file system library necessary for this call has not been installed or has not been opened.

sysErrParamErr The dirEntryIteratorP is not valid.

vfsErrFileBadRef The specified file reference is invalid.

vfsErrIsNotADirectory The specified file reference is valid, but does not point to a directory.

vfsErrNoFileSystem The VFS Manager cannot find an appropriate file system to handle the request.

Comments The directory to be enumerated must first be opened with VFSFileOpen in order to obtain a file reference. In order to obtain information on all entries in a directory you must make repeated calls to VFSDirEntryEnumerate inside a loop. Boundaries on the iteration are the defined constants vfsIteratorStart and vfsIteratorStop. Before the first call to VFSDirEntryEnumerate, dirEntryIteratorP should be initialized to vfsIteratorStart. Each iteration then changes the value pointed to by dirEntryIteratorP. When information on the last entry in the directory is returned, dirEntryIteratorP is set to vfsIteratorStop.

WARNING! Creating, renaming, or deleting any file or directory invalidates the enumeration. After any such operation, the enumeration will need to be restarted.

Example The following code excerpt illustrates how to use VFSDirEntryEnumerate.

```
FileInfoType info;
FileRef dirRef;
UInt32 dirIterator;
Char *fileName = MemPtrNew(256);  // should check for err

// open the directory first, to get the directory reference
// volRefNum must have already been defined
```
err = VFSFileOpen(volRefNum, "/", vfsModeRead, &dirRef);
if(err == errNone) {
    info.nameP = fileName;  // point to local buffer
    info.nameBufLen = 256;
    dirIterator = vfsIteratorStart
    while (dirIterator != vfsIteratorStop) {
        // Get the next file
        err = VFSDirEntryEnumerate (dirRef, &dirIterator,
            &info);
        if (err == errNone) {
            // Do something with the directory entry information
            // Pull the attributes from info.attributes
            // The file name is in fileName
            } else {
                // handle error, possibly by breaking out of the
                loop
            }
        } else {
            // handle directory open error here
        }
        MemPtrFree(fileName);
    }
}

Compatibility Implemented only if the VFS Manager Feature Set is present.

VFSExportDatabaseToFile

Purpose Save the specified database to a .pdb or .prc file on an external
storage card.

Declared In VfsMgr.h

Prototype Err VFSExportDatabaseToFile (UInt16 volRefNum,
const Char *pathNameP, UInt16 cardNo,
LocalID dbID)

Parameters -> volRefNum Volume on which the destination file should be
created.

-> pathNameP Pointer to the complete path and name of the
destination file to be created.
VFS Manager Functions

-> cardNo
Card number on which the .pdb or .prc being exported resides.

-> dbID
ID of the database being exported.

Result
Returns one of the following error codes:

errNone
No error

expErrNotEnoughPower
There is insufficient battery power to perform the database export operation.

vfsErrBadName
The path name specified in pathNameP is not valid.

Comments
This utility function exports a database from main memory to a .pdb or .prc file on an external storage card. This function is the opposite of VFSImportDatabaseFromFile. It first creates the file specified in the pathNameP parameter with VFSFileCreate. After opening the file the Exchange Manager function ExgDBWrite is called with an internal callback function for exporting the file from the Data Manager. The Exchange Manager makes repeated calls to this callback function, which receives the data back in blocks. Once all the data has been exported, VFS Manager closes the file.

This function is used, for example, to copy applications from main memory to a storage card.

Compatibility
Implemented only if the VFS Manager Feature Set is present.

See Also
VFSExportDatabaseToFileCustom, VFSFileWrite, VFSImportDatabaseFromFile

VFSExportDatabaseToFileCustom

Purpose
Saves the specified database to a .pdb or .prc file on an external storage card. This function differs from
**Virtual File System Manager**  
**VFS Manager Functions**

**VFSExportDatabaseToFile** in that it allows you to track the progress of the export operation.

**Declared In**  
VfsMgr.h

**Prototype**  
Err VFSExportDatabaseToFileCustom  
(UINT16 volRefNum, const Char *pathNameP,  
UINT16 cardNo, LocalID dbID,  
VFSExportProcPtr exportProcP, void *userDataP)

**Parameters**

- **-> volRefNum**  
  Volume on which the destination file should be created.

- **-> pathNameP**  
  Pointer to the complete path and name of the destination file to be created.

- **-> cardNo**  
  Card number on which the .pdb or .prc being exported resides.

- **-> dbID**  
  ID of the database being exported.

- **-> exportProcP**  
  User-defined callback function that tracks the progress of the export. This function should allow the user to cancel the export. Pass NULL if you don’t have a progress callback function. See **VFSExportProcPtr** for the requirements of this function.

- **-> userDataP**  
  Pointer to any data you want to pass to the callback function specified in exportProcP. This information is not used internally by the VFS Manager. Pass NULL if you don’t have a progress callback function or if that function doesn’t need any such data.

**Result**  
Returns one of the following error codes:

- **errNone**  
  No error.

- **expErrNotEnoughPower**  
  There is insufficient battery power to perform the database export operation.
vfsErrBadName  The path name specified in pathNameP is not valid.

This function can also return any error code other than errNone produced by your callback function.

Comments  This function is similar to VFSExportDatabaseToFile in that it exports a database from main memory to a .pdb or .prc file on an external storage card. It extends the functionality by allowing you to specify a callback function that tracks the progress of the export. It first creates the file specified in the pathNameP parameter with VFSFileCreate. After opening the file, the Exchange Manager function ExgDBWrite is called with an internal callback function for exporting the file from the Data Manager. Exchange Manager makes repeated calls to this function, which receives the data back in blocks. The progress tracker, if one has been specified, is also called every time a new chunk of data is passed back. Once all the data has been exported, the VFS Manager closes the file.

This function is used, for example, to copy applications from main memory to a storage card.

Compatibility  Implemented only if the VFS Manager Feature Set is present.

See Also  VFSExportDatabaseToFile, VFSFileWrite, VFSImportDatabaseFromFileCustom

VFSFileClose

Purpose  Closes a file or directory that has been opened with VFSFileOpen.

Declared In  VfsMgr.h

Prototype  Err VFSFileClose (FileRef fileRef)

Parameters  -> fileRef  File reference number returned from VFSFileOpen.

Result  Returns one of the following error codes:
Virtual File System Manager
VFS Manager Functions

errNone No error.
expErrNotOpen The file system library necessary for this call has not been installed or has not been opened.
vfsErrFileBadRef The specified file reference is invalid.

Compatibility Implemented only if the VFS Manager Feature Set is present.

VFSFileCreate

Purpose Create a file. This function cannot be used to create a directory; use VFSDirCreate instead.

Declared In VfsMgr.h

Prototype Err VFSFileCreate(UInt16 volRefNum, const Char *pathNameP)

Parameters -> volRefNum Reference number of the volume on which to create the file. This volume reference number is returned from VFSVolumeEnumerate.

-> pathNameP Pointer to the full path of the file to be created. All parts of the path, excluding the filename, must already exist.

Result Returns one of the following error codes:

errNone No error.
expErrNotOpen The file system library necessary for this call has not been installed or has not been opened.
vfsErrBadName The pathNameP is invalid.
vfsErrFileAlreadyExists A file with this name already exists in this location.
Virtual File System Manager
VFS Manager Functions

vfsErrNoFileSystem
The VFS Manager cannot find an appropriate file system to handle the request.

vfsErrVolumeBadRef
The volume has not been mounted.

vfsErrVolumeFull
There is not enough space left on the volume.

Comments
It is the responsibility of the file system library to ensure that all filenames are translated into a format that is compatible with the native format of the file system, such as the 8.3 convention for a FAT file system without long filename support. See Naming Files in the Expansion chapter of the Palm OS Programmer’s Companion, vol. I for a description of how to construct a valid path.

This function does not open the file. Use VFSFileOpen to open the file.

This function should not be used to create a directory. To create a directory use VFSDirCreate.

Compatibility
Implemented only if the VFS Manager Feature Set is present.

See Also
VFSFileDelete

VFSFileDBGetRecord

Purpose
Load a record from an opened .pdb file on an external card into the storage heap.

Declared In
VfsMgr.h

Prototype
Err VFSFileDBGetRecord (FileRef ref,
UInt16 recIndex, MemHandle *recHP,
UInt8 *recAttrP, UInt32 *uniqueIDP)

Parameters
-> ref
The file reference returned from VFSFileOpen. Note that the open file must be a .pdb file.
Virtual File System Manager
VFS Manager Functions

-> recIndex  The index of the record to load.

<- recHP  Pointer to the record data’s handle in the storage heap. If NULL is returned in this parameter there is either no data in this field or an error occurred reading this data from the file. If the handle is not NULL, you must dispose of the allocated handle using MemHandleFree.

<- recAttrP  Pointer to the attributes of the record. The values returned are identical to the attributes returned from DmRecordInfo. See Record Attribute Constants in the Data and Resource Manager chapter for a description of each attribute. Pass NULL for this parameter if you do not want to retrieve this information.

<- uniqueIDP  Pointer to the unique identifier for this record. Pass NULL for this parameter if you do not want to retrieve this information.

Result  Returns one of the following error codes:

errNone  No error.

dmErrIndexOutOfRange  The recIndex is out of range.

dmErrNotRecordDB  The file referenced by ref is not a record database.

memErrNotEnoughSpace  There is not enough space in memory for the requested record entry.

sysErrParamErr  A NULL value was passed in for the recHP, recAttrP, and uniqueIDP parameters.

vfsErrBadData  The local offsets (localChunkID) from the top of the .pdb to the start of the raw record data for this entry are out of order.

Comments  This function is analogous to DmGetRecord but works with files on an external card rather than databases in main memory. This
function allocates a handle of the appropriate size from the storage heap and returns it in the `recHP` parameter. The caller is responsible for freeing this memory, using `MemHandleFree`, when it is no longer needed.

**NOTE:** This function is not efficient for multiple accesses and should be used sparingly.

**Compatibility**
Implemented only if the VFS Manager Feature Set is present.

**See Also**
VFSFileReadData

## VFSFileDBGetResource

**Purpose**
Load a resource into the storage heap from an opened `.prc` file.

**Declared In**
VfsMgr.h

**Prototype**
```c
Err VFSFileDBGetResource (FileRef ref, DmResType type, DmResID resID, MemHandle *resHP)
```

**Parameters**
- `-> ref`    The file reference returned from `VFSFileOpen`. Note that the open file must be a `.prc` file.
- `-> type`   The type of resource to load. See the Data and Resource Manager chapter for more information on resources.
- `-> resID`  The ID of resource to load.
- `<- resHP`  Pointer to the resource data handle that was loaded into memory.

**Result**
Returns one of the following error codes:
- `errNone`  No error.
dmErrNotResourceDB
The file referenced by ref is not a resource database.

dmErrResourceNotFound
The requested resource was not found.

memErrNotEnoughSpace
There is not enough space in memory for the requested resource entries.

sysErrParamErr resHP is NULL.

Comments
This function locates the specified resource in the open .prc file. See the Palm OS File Format Specification for more information on the layout of .prc files.

Once the resource is found, VFSFileDBGetResource allocates a handle of the appropriate size in the storage heap and reads it into memory. The handle to this memory location is returned through the resHP parameter. The caller is responsible for freeing this memory, using MemHandleFree, when it is no longer needed.

NOTE: This function is not efficient for multiple accesses and should be used sparingly.

Compatibility
Implemented only if the VFS Manager Feature Set is present.
**VFSFileDBInfo**

**Purpose** Get information about a database represented by an open .prc or .pdb file.

**Declared In** VfsMgr.h

**Prototype**

```
Err VFSFileDBInfo (FileRef ref, Char *nameP,
UInt16 *attributesP, UInt16 *versionP,
UInt32 *crDateP, UInt32 *modDateP,
UInt32 *bckUpDateP, UInt32 *modNumP,
MemHandle *appInfoHP, MemHandle *sortInfoHP,
UInt32 *typeP, UInt32 *creatorP,
UInt16 *numRecordsP)
```

**Parameters**

- **-> ref** The file reference returned from VFSFileOpen. Note that the open file must be a .prc or .pdb file.

- **<- nameP** Pointer to a 32-byte character array in which the database name is returned. Pass NULL for this parameter if you do not want to retrieve the database name.

- **<- attributesP** Pointer to the database attributes stored in the file. The values returned are identical to the attributes returned from DmDatabaseInfo. See the Database Attribute Constants section for a description of each attribute. Pass NULL for this parameter if you do not want to retrieve the database’s attributes.

- **<- versionP** Pointer to the application-specific version number of the database. The default version number is zero (0). Pass NULL for this parameter if you do not want to retrieve the version number.

- **<- crDateP** Pointer to the date the database was created, expressed in seconds since midnight (00:00:00) January 1, 1904. Pass NULL for this parameter if you do not want to retrieve the creation date.
<- modDateP  Pointer to the date the database was last modified, expressed in seconds since midnight (00:00:00) January 1, 1904. A database’s modification date is updated only if a change has been made to the database when it is opened with write access. Pass NULL for this parameter if you do not want to retrieve the database’s modification date.

<- bckUpDateP  Pointer to the date the database was last backed up, expressed in seconds since midnight (00:00:00) January 1, 1904. Pass NULL for this parameter if you do not want to retrieve the database’s backup date.

<- modNumP  Pointer to the number of times the database was modified. This number is updated every time a record is added, modified, or deleted. Pass NULL for this parameter if you do not want to retrieve the modification count.

<- appInfoHP  Pointer to the application info block handle. If NULL is returned in this parameter, either there is no data in this field or an error occurred reading this data from the file. If a value other than NULL is returned, you must dispose of the allocated handle using MemHandleFree. If you do not want to retrieve the application info block, pass NULL for this parameter.

<- sortInfoHP  Pointer to the sort info block handle. If NULL is returned in this parameter, either there is no data in this field or an error occurred reading this data from the file. If a value other than NULL is returned, you must dispose of the allocated handle using MemHandleFree. Pass NULL for this parameter if you do not want to retrieve the sort info block handle.
<- typeP  

Pointer to the type of database as it was created. This may be a user-defined database type or a database type defined by the Palm OS. Some of the more common database types returned here are:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'appl'</td>
<td>Standard Palm™ application (resource database)</td>
</tr>
<tr>
<td>'libr'</td>
<td>Standard shared library</td>
</tr>
<tr>
<td>'libf'</td>
<td>File system shared library</td>
</tr>
<tr>
<td>'libs'</td>
<td>Slot driver shared library</td>
</tr>
<tr>
<td>'data'</td>
<td>Standard Palm data file (record database)</td>
</tr>
</tbody>
</table>

Pass NULL for this parameter if you do not want to retrieve the database’s type.

<- creatorP  

Pointer to the database’s creator. Pass NULL for this parameter if you do not want to retrieve this information.

<- numRecordsP  

Pointer to the number of records in the database. Pass NULL for this parameter if you do not want to retrieve this information.

**Result**  

Returns one of the following error codes:

- **errNone**  
  No error

- **memErrNotEnoughSpace**  
  There is not enough space in memory for the database header.

- **vfsErrBadData**  
  The file referenced by the ref parameter is too small to contain a database header, or the database header is corrupted.
Comments  This function is analogous to DmDatabaseInfo, but it works with files on an external card rather than with databases in main memory. See the Palm OS File Format Specification for a description of the header block in .prc and .pdb files.

Compatibility  Implemented only if the VFS Manager Feature Set is present.

See Also  VFSFileGetAttributes, VFSFileGetDate

VFSFileDelete

Purpose  Deletes a closed file or directory.

Declared In  VfsMgr.h

Prototype  Err VFSFileDelete(UInt16 volRefNum, const Char *pathNameP)

Parameters
  -> volRefNum  Volume reference number returned from VFSVolumeEnumerate.
  -> pathNameP  Pointer to the full path of the file or directory to be deleted.

Result  Returns one of the following error codes:
  errNone  No error.
  expErrNotOpen  The file system library necessary for this call has not been installed or has not been opened.
  vfsErrBadName  The path name specified in pathNameP is not valid.
  vfsErrDirNotEmpty  The directory being deleted is not empty.
  vfsErrFileStillOpen  The file is still open.
  vfsErrFileNotFound  The file could not be found.
vfsErrFilePermissionDenied
The requested permissions could not be granted.

vfsErrNoFileSystem
The VFS Manager cannot find an appropriate file system to handle the request.

vfsErrVolumeBadRef
The volume has not been mounted.

Compatibility Implemented only if the VFS Manager Feature Set is present.

**VFSFileEOF**

**Purpose** Get end-of-file status for an open file. This function only operates on files and cannot be used with directories.

**Declared In** VfsMgr.h

**Prototype** Err VFSFileEOF (FileRef fileRef)

**Parameters**
- -> fileRef File reference returned from VFSFileOpen.

**Result** Returns one of the following error codes:
- errNone No error. File pointer is not at end of the file.
- vfsErrFileEOF The file pointer is at the end of file.
- expErrNotOpen The file system library necessary for this call has not been installed or has not been opened.
- vfsErrFileBadRef The specified file reference is invalid.
- vfsErrIsADirectory The specified file reference points to a directory instead of a file. This is an invalid operation on a directory.
Virtual File System Manager
VFS Manager Functions

vfsErrNoFileSystem
The VFS Manager cannot find an appropriate file system to handle the request.

Compatibility
Implemented only if the VFS Manager Feature Set is present.

VFSFileGetAttributes

Purpose
Obtain the attributes of an open file or directory.

Declared In
VfsMgr.h

Prototype
Err VFSFileGetAttributes (FileRef fileRef,
UInt32 *attributesP)

Parameters
-> fileRef File reference returned from VFSFileOpen.
<- attributesP Pointer to the attributes associated with the file or directory.
See File and Directory Attributes for a list of values that can be returned through this parameter.

Result
Returns one of the following error codes:
errNone No error
expErrNotOpen The file system library necessary for this call has not been installed or has not been opened.
vfsErrFileBadRef The specified file reference is invalid.
vfsErrNoFileSystem The VFS Manager cannot find an appropriate file system to handle the request.

Compatibility
Implemented only if the VFS Manager Feature Set is present.

See Also
VFSFileDBInfo, VFSFileGetDate, VFSFileSetAttributes
**VFSFileGetDate**

**Purpose**  
Obtain the dates on an open file or directory.

**Declared In**  
VfsMgr.h

**Prototype**  
Err VFSFileGetDate (FileRef fileRef, UInt16 whichDate, UInt32 *dateP)

**Parameters**  
-> fileRef  
File reference returned from VFSFileOpen.

-> whichDate  
Specifies which date—creation, modification, or last access—you want. Supply one of the following values:

- vfsFileDateCreated
- vfsFileDateModified
- vfsFileDateAccessed

Note that not all file systems are required to support the above dates. If the supplied date type is not supported by the file system, VFSFileGetDate returns expErrUnsupportedOperation.

<- dateP  
Pointer to the requested date. This field is expressed in the standard Palm OS date format — the number of seconds since midnight (00:00:00) January 1, 1904.

**Result**  
Returns one of the following error codes:

- errNone  
No error.

- expErrNotOpen  
The file system library necessary for this call has not been installed or has not been opened.

- expErrUnsupportedOperation  
The specified date type is not supported by the underlying file system.
Virtual File System Manager
VFS Manager Functions

vfsErrFileBadRef
   The specified file reference is invalid.

sysErrParamErr
   The whichDate parameter is not one of the defined constants.

Compatibility
   Implemented only if the VFS Manager Feature Set is present.

See Also
   VFSFileDBInfo, VFSFileGetAttributes, VFSFileSetDate

VFSFileOpen

Purpose
   Opens a file or directory and returns a reference for it.

Declared In
   VfsMgr.h

Prototype
   Err VFSFileOpen (UInt16 volRefNum,
                   const Char *pathNameP, UInt16 openMode,
                   FileRef *fileRefP)

Parameters
   - > volRefNum   The volume reference number returned from
                   VFSVolumeEnumerate.
   - > pathNameP   Pointer to the full path of the file or directory
                   to be opened. This must be a valid path. It cannot
                   be empty and can not contain null characters.
                   The format of the pathname should match what
                   the underlying file system supports. See
                   “Naming Files” in Chapter 7, “Expansion,” of
                   the Palm OS Programmer’s Companion, vol. I for
                   a description of how to construct a valid path.
   - > openMode    Mode to use when opening the file. See the
                   Open Mode Constants section for a list of
                   accepted modes.
<- fileRefP     Pointer to the opened file or directory reference which is supplied to various other VFSFile... operations. This value is filled in on return.

Result  Returns one of the following error codes:

errNone          No error.
expErrCardReadOnly The open mode requested includes write access but the file is read-only.
expErrNotOpen    The file system library necessary for this call has not been installed or has not been opened.
vfsErrBadName    The pathNameP parameter is invalid.
vfsErrFileNotFound The specified file or directory could not be found.
vfsErrFilePermissionDenied The file cannot be opened in the requested open mode, or it has already been opened with vfsModeExclusive.
vfsErrVolumeBadRef The specified volume has not been mounted.

Compatibility  Implemented only if the VFS Manager Feature Set is present.

See Also  VFSFileClose, VFSDirEntryEnumerate
VFSFileRead

Purpose: Read data from a file into the dynamic heap. This function only operates on files and cannot be used with directories; use VFSDirEntryEnumerate to explore the contents of a directory.

Declared In: VfsMgr.h

Prototype: Err VFSFileRead (FileRef fileRef, UInt32 numBytes, void *bufP, UInt32 *numBytesReadP)

Parameters:
- fileRef: File reference returned from VFSFileOpen.
- numBytes: Number of bytes to read.
- bufP: Pointer to the destination chunk where the data is to be stored. This can be a pointer to any writable memory.
- numBytesReadP: Pointer to an unsigned integer that reflects the number of bytes actually read. This value is set on return and does not need to be initialized. If no bytes are read the value is set to zero. Pass NULL for this parameter if you do not need to know how many bytes were read.

Result: Returns one of the following error codes:
- errNone: No error.
- expErrNotOpen: The file system library necessary for this call has not been installed or has not been opened.
- vfsErrFileBadRef: The specified file reference is invalid.
- vfsErrFileEOF: The end of the file has been reached.
- vfsErrFilePermissionDenied: Read permission is not enabled for this file.
vfsErrIsADirectory
The specified file reference is for a directory instead of a file. This is an invalid operation on a directory.

vfsErrNoFileSystem
The VFS Manager cannot find an appropriate file system to handle the request.

Comments
The file system does not use DmWrite and cannot be used to read data into the storage heap.

Compatibility
Implemented only if the VFS Manager Feature Set is present.

See Also
VFSFileReadData, VFSFileWrite, VFSImportDatabaseFromFile

VFSFileReadData

Purpose
Read data from a file into a chunk of memory in the storage heap. This function only operates on files and cannot be used with directories; use VFSDirEntryEnumerate to explore the contents of a directory.

Declared In
VfsMgr.h

Prototype
Err VFSFileReadData (FileRef fileRef, UInt32 numBytes, void *bufBaseP, UInt32 offset, UInt32 *numBytesReadP)

Parameters
-> fileRef File reference returned in VFSFileOpen.
-> numBytes Number of bytes to read.
<- bufBaseP Pointer to the destination chunk in the storage heap where the data is to be stored. This pointer must be obtained through the appropriate call to the Memory Manager API.
-> offset Offset, in bytes, within the bufBaseP chunk where the data is to be written.
Virtual File System Manager
VFS Manager Functions

<- numBytesReadP

Pointer to an unsigned integer that reflects the number of bytes actually read. This value is set on return and does not need to be initialized. If no bytes are read, the value is set to zero. Pass NULL for this parameter if you do not need to know how many bytes were read.

Result
Returns one of the following error codes:

errNone No error.
expErrNotOpen The file system library necessary for this call has not been installed or has not been opened.
vfsErrFileBadRef The specified file reference is invalid.
vfsErrFileEOF The end of the file has been reached.
vfsErrFilePermissionDenied Read permission is not enabled for this file.
vfsErrIsADirectory The specified file reference is for a directory instead of a file. This is an invalid operation on a directory.
vfsErrNoFileSystem The VFS Manager cannot find an appropriate file system to handle the request.

Comments
When data is read from an external card with VFSFileReadData, it is copied into a chunk of memory in the storage heap. This chunk must be allocated by the application before the call to VFSFileReadData. This function calls DmWrite to put the data in the storage heap.

Compatibility
Implemented only if the VFS Manager Feature Set is present.

See Also
VFSFileRead, VFSFileWrite
VFSFileRename

Purpose
Rename a closed file or directory. This function cannot be used to move a file to another directory within the file system.

Declared In
VfsMgr.h

Prototype
Err VFSFileRename (UInt16 volRefNum, const Char *pathNameP, const Char *newNameP)

Parameters
- volRefNum Volume reference number returned from VFSVolumeEnumerate.
- pathNameP Pointer to the full path of the file or directory to be renamed.
- newNameP Pointer to the new filename. Note that this is the name of the file only and does not include the path to the file.

Result
Returns one of the following error codes:
- errNone No error.
- expErrNotOpen The file system library necessary for this call has not been installed or has not been opened.
- vfsErrBadName The name provided in either pathNameP or newNameP is invalid. This is also returned if the string pointed to by newNameP is a path, rather than a filename.
- vfsErrFileAlreadyExists A file with the new name already exists in this location.
- vfsErrFileNotFoundException The source file could not be found.
- vfsErrFilePermissionDenied Write permission is not enabled for this file.
Virtual File System Manager
VFS Manager Functions

vfsErrNoFileSystem
The VFS Manager cannot find an appropriate file system to handle the request.

vfsErrVolumeBadRef
The volume has not been mounted.

vfsErrVolumeFull
There is not enough space left on the volume.

Comments
WARNING! This function invalidates directory enumeration. You cannot continue enumerating files after renaming one of them with this function. If you need to operate on additional files in the directory, you must first restart the enumeration.

Example
Below is an example of how to use VFSFileRename. Note that the renamed file remains in the /PALM/Programs directory; VFSFileRename can’t be used to move files from one directory to another.
Virtual File System Manager
VFS Manager Functions

// volRefNum must have been previously defined; most likely,
// it was returned by VFSVolumeEnumerate

err = VFSFileRename(volRefNum, "/PALM/Programs/foo.prc",
                     "bar.prc");
if (err != errNone) {
    // handle error...
}

Compatibility
Implemented only if the VFS Manager Feature Set is present.

VFSFileResize

Purpose
Change the size of an open file. This function only operates on files
and cannot be used with directories.

Declared In
VfsMgr.h

Prototype
Err VFSFileResize (FileRef fileRef,
                   UInt32 newSize)

Parameters
-> fileRef      File reference returned from VFSFileOpen.
-> newSize      The desired new size of the file. This can be
                larger or smaller than the current file size.

Result
Returns one of the following error codes:
errNone        No error.
expErrNotOpen  The file system library necessary for this call
                has not been installed or has not been opened.
vfsErrFileBadRef  The specified file reference is invalid.
vfsErrIsADirectory
                The specified file reference points to a directory
                instead of a file. This is an invalid operation on
                a directory.
Virtual File System Manager
VFS Manager Functions

vfsErrNoFileSystem
The VFS Manager cannot find an appropriate file system to handle the request.

vfsErrVolumeFull
There is not enough space left on the volume.

Comments
The location of the file pointer is undefined after a call to this function.

Compatibility
Implemented only if the VFS Manager Feature Set is present.

See Also
VFSFileSize

VFSFileSeek

Purpose
Set the position within an open file from which to read or write. This function only operates on files and cannot be used with directories.

Declared In
VfsMgr.h

Prototype
Err VFSFileSeek (FileRef fileRef, FileOrigin origin, Int32 offset)

Parameters
- fileRef
  File reference returned from VFSFileOpen.
- origin
  Origin to use when calculating the new position. The offset parameter indicates the desired new position relative to this origin, which can be one of the following:
    - vfsOriginBeginning
      The beginning of the file.
    - vfsOriginCurrent
      The current position within the file.
    - vfsOriginEnd
      The end of the file. Only negative offsets are allowed when origin is set to vfsOriginEnd.
VFSFileSetAttributes

Purpose
Change the attributes of an open file or directory.

Declared In
VfsMgr.h

Prototype
Err VFSFileSetAttributes (FileRef fileRef, UInt32 attributes)
Parameters

- fileRef: File reference returned from `VFSFileOpen`.
- attributes: Attributes to associate with the file or directory. See File and Directory Attributes for a list of values you can use when setting this parameter.

Result

Returns one of the following error codes:

- `errNone`: No error.
- `expErrNotOpen`: The file system library necessary for this call has not been installed or has not been opened.
- `sysErrParamErr`: One of the parameters is invalid.
- `vfsErrFileBadRef`: The specified file reference is invalid.
- `vfsErrNoFileSystem`: The VFS Manager cannot find an appropriate file system to handle the request.

Comments

**NOTE:** You cannot use this function to set the `vfsFileAttrDirectory` or `vfsFileAttrVolumeLabel` attributes. The `vfsFileAttrDirectory` is set when you call `VFSDirCreate`. The `vfsFileAttrVolumeLabel` is set when you call `VFSVolumeSetLabel`. This function may fail when setting other attributes, depending on the underlying file system.

Compatibility

Implemented only if the VFS Manager Feature Set is present.

See Also

`VFSFileGetAttributes`, `VFSFileSetDate`
VFSFileSetDate

Purpose Changes the dates on an open file or directory.

Declared In VfsMgr.h

Prototype Err VFSFileSetDate (FileRef fileRef, UInt16 whichDate, UInt32 date)

Parameters
-> fileRef File reference returned in VFSFileOpen.

-> whichDate Specifies which date—creation, modification, or last access—to modify. Supply one of the following values:

  vfsFileDateCreated
  vfsFileDateModified
  vfsFileDateAccessed

Note that not all file systems are required to support the above dates. If the supplied date type is not supported by the file system, VFSFileGetDate returns expErrUnsupportedOperation.

-> date The new date. This field should be expressed in the standard Palm OS date format — the number of seconds since midnight (00:00:00) January 1, 1904.

Result Returns one of the following error codes:

  errNone No error.

  expErrNotOpen The file system library necessary for this call has not been installed or has not been opened.

  expErrUnsupportedOperation The specified date type is not supported by the underlying file system.

  sysErrParamErr The whichDate parameter is not one of the defined constants.
Virtual File System Manager
VFS Manager Functions

vfsErrFileBadRef
The specified file reference is invalid.

vfsErrFilePermissionDenied
Write permission is not enabled for this file.

vfsErrNoFileSystem
The VFS Manager cannot find an appropriate file system to handle the request.

Compatibility
Implemented only if the VFS Manager Feature Set is present.

See Also
VFSFileGetDate, VFSFileSetAttributes

VFSFileSize

Purpose
Obtain the size of an open file. This function only operates on files and cannot be used with directories.

Declared In
VfsMgr.h

Prototype
Err VFSFileSize (FileRef fileRef, UInt32 *fileSizeP)

Parameters
-> fileRef File reference returned from VFSFileOpen.
<- fileSizeP Pointer to the size of the open file.

Result
Returns one of the following error codes:
errNone No error.
expErrNotOpen The file system library necessary for this call has not been installed or has not been opened.
vfsErrFileBadRef The specified file reference is invalid.
vfsErrIsADirectory The specified file reference points to a directory instead of a file. This is an invalid operation on a directory.
vfsErrNoFileSystem
The VFS Manager cannot find an appropriate
file system to handle the request.

Compatibility
Implemented only if the VFS Manager Feature Set is present.

See Also VFSFileResize, VFSFileTell, VFSVolumeSize

VFSFileTell

Purpose
Get the current position of the file pointer within an open file. This
function only operates on files and cannot be used with directories.

Declared In VfsMgr.h

Prototype
Err VFSFileTell (FileRef fileRef,
UInt32 *filePosP)

Parameters
-> fileRef File reference returned from VFSFileOpen.
<- filePosP Pointer to the current file position.

Result
Returns one of the following error codes:
errNone No error.
expErrNotOpen The file system library necessary for this call has not been installed or has not been opened.
vfsErrFileBadRef The specified file reference is invalid.
vfsErrIsADirectory The specified file reference points to a directory instead of a file. This is an invalid operation on a directory.
vfsErrNoFileSystem The VFS Manager cannot find an appropriate file system to handle the request.
Virtual File System Manager
VFS Manager Functions

Compatibility
Implemented only if the VFS Manager Feature Set is present.

See Also
VFSFileSeek, VFSFileSize

VFSFileWrite

Purpose
Write data to an open file. This function only operates on files and cannot be used with directories.

Declared In
VfsMgr.h

Prototype
Err VFSFileWrite (FileRef fileRef, UInt32 numBytes, const void *dataP, UInt32 *numBytesWrittenP)

Parameters
-> fileRef
File reference returned from VFSFileOpen.

-> numBytes
The number of bytes to write.

-> dataP
Pointer to the data that is to be written.

<- numBytesWrittenP
Pointer to an unsigned integer that reflects the number of bytes actually written. This value is set on return and does not need to be initialized. If no bytes are written the value is set to zero. Pass NULL for this parameter if you do not need to know how many bytes were written.

Result
Returns one of the following error codes:

errNone
No error.

expErrNotOpen
The file system library necessary for this call has not been installed or has not been opened.

vfsErrFileBadRef
The specified file reference is invalid.

vfsErrFilePermissionDenied
Write permission is not enabled for this file.
vfsErrIsADirectory
The specified file reference points to a directory
instead of a file. This is an invalid operation on
a directory.

vfsErrNoFileSystem
The VFS Manager cannot find an appropriate
file system to handle the request.

vfsErrVolumeFull
There is not enough space left on the volume.

Compatibility
Implemented only if the VFS Manager Feature Set is present.

See Also
VFSExportDatabaseToFile,
VFSExportDatabaseToFileCustom, VFSFileRead,
VFSFileReadData

VFSGetDefaultDirectory

Purpose
Determine the default location on the given volume for files of a
particular type.

Declared In
VfsMgr.h

Prototype
Err VFSGetDefaultDirectory (UInt16 volRefNum,
const Char *fileTypeStr, Char *pathStr,
UInt16 *bufLenP)

Parameters
-> volRefNum Volume reference number returned from
VFSVolumeEnumerate.

-> fileTypeStr Pointer to the requested file type, as a null-
terminated string. The file type may either be a
MIME media type/subtype pair, such as
“image/jpeg”, “text/plain”, or “audio/basic”;
or a file extension, such as “.jpeg.”

<- pathStr Pointer to the buffer which receives the default
directory path for the requested file type.
Virtual File System Manager
VFS Manager Functions

<- bufLenP

Pointer to the size of the path. Set this to the size of pathStr buffer on input. Reflects the number of bytes copied to pathStr on output.

Result

Returns one of the following error codes:

errNone No error.
vfsErrBadName There is no default directory registered for the requested file type.
vfsErrBufferOverflow A match was found, but the pathStr buffer is too small to hold the resulting path string. A partial path is returned in pathStr.
vfsErrFileNotFound No match was found for the specified volume. The error could have occurred with either the media type specified for this volume or the file type requested.

Comments

This function returns the complete path to the default directory registered for the specified file type. A default directory can be registered for each type of media supported. The directory should be registered under media and file type. Note that this directory is typically a “root” directory for the file type; any subdirectories under this root directory should also be searched for files of the appropriate type.

This function can be used by an image viewer application, for example, to find the directory containing images without having to know what type of media the volume was on. This could be “/DCIM”, “/images”, or something else depending on the type of media.

Compatibility

Implemented only if the VFS Manager Feature Set is present.

See Also

VFSDirEntryEnumerate, VFSRegisterDefaultDirectory, VFSUnregisterDefaultDirectory
VFSImportDatabaseFromFile

**Purpose**
Creates a database from a .pdb or .prc file on an external storage card.

**Declared In**
VfsMgr.h

**Prototype**
Err VFSImportDatabaseFromFile (UInt16 volRefNum, const Char *pathNameP, UInt16 *cardNoP, LocalID *dbIDP)

**Parameters**
- `volRefNum` Volume on which the source file resides.
- `pathNameP` Pointer to the full path and name of the source file.
- `cardNoP` Pointer to a variable that receives the card number of the newly-created database. If the database already resides in the storage heap, the card number of the existing database is returned along with the error dmErrAlreadyExists.
- `dbIDP` Pointer to a variable that receives the database ID of the new database. If the database already resides in the storage heap, the database ID of the existing database is returned along with the error dmErrAlreadyExists.

**Result**
Returns one of the following error codes:
- `errNone` No error.
- `dmErrAlreadyExists` The .prc or .pdb file already exists in the storage heap. In this case the `cardNoP` and `dbIDP` are set to point to the existing file.
- `expErrNotEnoughPower` There is insufficient battery power to complete the requested operation.
VFSManager Functions

vfsErrBadName

The path name specified in pathNameP is not valid.

Comments

This utility function imports a .pdb or .prc file resident on an external storage card into a new database in the storage heap. It first calls VFSFileOpen to open the file specified in pathNameP. Assuming that a corresponding .prc or .pdb does not already exist in the storage heap, VFSImportDatabaseFromFile calls the Exchange Manager function ExgDBRead with an internal callback function for importing a file to the Data Manager. The Exchange Manager makes repeated calls to this function, which passes the data back in blocks. Once the file has been successfully imported, the owner (the imported file, if it’s an executable, or the associated application if it is not) is sent a sysAppLaunchCmdSyncNotify launch code to make it aware of the new database.

This function doesn’t provide any progress indication to the user. If you need to provide feedback to the user as the file import progresses, use VFSImportDatabaseFromFileCustom instead.

This function is used, for example, to copy applications from a storage card to main memory.

Compatibility

Implemented only if the VFS Manager Feature Set is present.

See Also

VFSExportDatabaseToFile, VFSFileRead

VFSImportDatabaseFromFileCustom

Purpose

Create a database from the specified .pdb or .prc file on an external storage card. This function differs from
VFSImportDatabaseFromFile in that it allows you to track the progress of the import operation.

**Declared In**  
VfsMgr.h

**Prototype**  
Err VFSImportDatabaseFromFileCustom  
(Uint16 volRefNum, const Char *pathNameP,  
Uint16 *cardNoP, LocalID *dbIDP,  
VFSImportProcPtr importProcP, void *userDataP)

**Parameters**

- **volRefNum**  
Volume on which the source file resides.

- **pathNameP**  
Pointer to the full path and name of the source file.

- **cardNoP**  
Pointer to the variable that receives the card number of the newly-created database. If the database already resides in the storage heap, the card number of the existing database is returned along with the error dmErrAlreadyExists.

- **dbIDP**  
Pointer to the variable that receives the database ID of the new database. If the database already resides in the storage heap, the database ID of the existing database is returned along with the error dmErrAlreadyExists.

- **importProcP**  
User-defined callback function that tracks the progress of the import. This function should allow the user to cancel the import. Pass NULL if you don’t have a progress callback function. See VFSImportProcPtr for the requirements of this function.
Virtual File System Manager

VFS Manager Functions

-> userDataP  Pointer to any data you want to pass to the callback function specified in importProcP. This information is not used internally by the VFS Manager. Pass NULL if you don’t have a progress callback function, or if that function doesn’t need any such data.

Result  Returns one of the following error codes:

errNone  No error

vfsErrBadName  The path name specified in pathNameP is not valid.

expErrNotEnoughPower  The power required to import a database is not available.

dmErrAlreadyExists  The .prc or .pdb file already exists in main memory. In this case the cardNoP and dbIDP are set to point to the existing file.

Comments  This function is similar to VFSImportDatabaseFromfile in that it imports a .pdb or .prc file on an external storage card into a new database on the storage heap. It extends the functionality by allowing you to specify a callback function that tracks the progress of the export. It first calls VFSFileOpen to open the file specified in pathNameP. If a corresponding .prc or .pdb does not already exist in main memory, it calls the Exchange Manager function ExgDBRead with an internal callback function for importing the file from the Data Manager. The Exchange Manager makes repeated calls to this function, which receives the data back in blocks. The progress tracker, if one has been specified, is also called every time a new chunk of data is passed back. Once the file has been successfully imported, the owner (the imported file, if it’s an executable, or the associated application if it is not) is sent a sysAppLaunchCmdSyncNotify launch code to make it aware of the new database.

This function is used, for example, to copy applications from a storage card to main memory.
Compatibility  Implemented only if the VFS Manager Feature Set is present.

See Also  VFSFileRead, VFSExportDatabaseToFileCustom

**VFSInstallFSLib**

**Purpose**  Install a file system library so that the VFS Manager can use it.

**Declared In**  VfsMgr.h

**Prototype**  Err VFSInstallFSLib (UInt32 creator, UInt16 *fsLibRefNumP)

**Parameters**

-> creator  Creator ID of the database containing the file system library to be installed.

<- fsLibRefNumP  Pointer to the reference number for the newly installed file system library. Supply NULL for this parameter if you don’t need the library reference number.

**Result**  If the file system library was loaded and installed without error, errNone is returned. Any error generated by the underlying file system while opening the file system library or determining its type will be returned from VFSInstallFSLib. Other errors that can be generated during the file system library installation process include:

expErrIncompatibleAPIVer  The file system library has an incompatible API version.

expErrNotOpen  The file system library necessary for this call has not been installed or has not been opened.

memErrNotEnoughSpace, memErrChunkNotLocked, or memErrChunkLocked  A memory problem occurred while inserting the library reference into the list of installed libraries.
**Virtual File System Manager**  
*VFS Manager Functions*

sysErrLibNotFound, sysErrNoFreeRAM,  
sysErrNoFreeLibSlots (or some other  
error returned from the library’s install entry  
point)  
An error occurred while loading the library.

**Comments**  This function calls **SysLibLoad** to load the file system library into  
the library table. Once loaded the appropriate file system is asked to  
open the library. At boot time **VFSInstallFSLib** is called  
internally by the Expansion Manager to load all installed file system  
libraries and initialize them for use.  
**VFSInstallFSLib** is not normally called by applications.

**Compatibility**  Implemented only if the **VFS Manager Feature Set** is present.

**See Also**  **VFSRemoveFSLib**

### **VFSRegisterDefaultDirectory**

**Purpose**  Registers a specific directory as the default location for files of a  
given type on a particular kind of external storage card. This  
function is generally called by a slot driver for files and media types  
that are supported by that slot driver.

**Declared In**  **VfsMgr.h**

**Prototype**  
```
Err VFSRegisterDefaultDirectory  
(const Char *fileTypeStr, UInt32 mediaType,  
const Char *pathStr)
```

**Parameters**  -> **fileTypeStr**  Pointer to the file type to register. This is a null-  
terminated string that can either be a MIME  
media type/subtype pair, such as “image/  
jpeg”, “text/plain”, or “audio/basic”; or a file  
extension, such as “.jpeg”.
Virtual File System Manager

VFS Manager Functions

-> mediaType Type of card media for which the default directory is being associated. See Defined Media Types in the Expansion Manager chapter for the list of accepted values.

-> pathStr Pointer to the default directory path to be associated with the specified file type. This string must be null-terminated, and must be the full path to the directory.

Result Returns one of the following error codes:

errNone No error.

sysErrParamErr Either the fileTypeStr parameter is NULL or the pathStr parameter is NULL.

vfsErrFileAlreadyExists A default directory has already been registered for this file type on the specified card media type.
**Virtual File System Manager**

**VFS Manager Functions**

**Comments**
This function first verifies that a default directory has not already been registered for the specified combination of file type and media type, and returns `vfsErrFileAlreadyExists` if one has been registered. To change an existing entry in the registry, you must first remove the existing entry with a call to `VFSUnregisterDefaultDirectory` before re-registering it with `VFSRegisterDefaultDirectory`.

The specified directory registered for a given file type is intended to be the “root” default directory. If a given default directory has one or more subdirectories, applications should also search those subdirectories for files of the appropriate type.

**Compatibility**
Implemented only if the VFS Manager Feature Set is present.

**See Also**
`VFSGetDefaultDirectory`

**VFSRemoveFSLib**

**Purpose**
Remove a file system library from the library table, so that the VFS Manager can no longer use it.

**Declared In**
VfsMgr.h

**Prototype**
`Err VFSRemoveFSLib (UInt16 fsLibRefNum)`

**Parameters**
- `fsLibRefNum` Library reference number of the file system library to remove from the library table.

**Result**
Returns one of the following error codes:

- `errNone` No error.
- `expErrNotOpen` The file system library necessary for this call has not been installed or has not been opened.
- `vfsErrNoFileSystem` VFS Manager can not find the file system specified in `fsLibRefNum`.
Comments  This function is not normally called by applications. It unmounts any volumes that the specified file system may have mounted. It then closes the library and removes it from the library table with SysLibRemove.

Compatibility  Implemented only if the VFS Manager Feature Set is present.

**VFSUnregisterDefaultDirectory**

**Purpose**  Sever the association between a particular file type and a default directory for a given type of card media.

**Declared In**  VfsMgr.h

**Prototype**  Err VFSUnregisterDefaultDirectory
(const Char *fileTypeStr, UInt32 mediaType)

**Parameters**

- -> fileTypeStr  Pointer to the file type with which the default directory is associated. This is a null-terminated string that can either be a MIME media type/subtype pair, such as “image/jpeg”, “text/plain”, or “audio/basic”; or a file extension, such as “.jpeg”.

- -> mediaType  Type of card media for which the default directory is associated. See Defined Media Types in the Expansion Manager chapter for the list of accepted values.

**Result**  Returns one of the following error codes:

- errNone  No error.

- sysErrParamErr  The fileTypeStr parameter is NULL.

- vfsErrFileNotFound  A default directory could not be found in the registry for the specified file and media type.
**Virtual File System Manager**

**VFS Manager Functions**

<table>
<thead>
<tr>
<th>Comments</th>
<th>NOTE: Caution is advised when using this function, since you may remove another application’s registration, causing data to mysteriously disappear from those applications.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compatibility</td>
<td>Implemented only if the <a href="#">VFS Manager Feature Set</a> is present.</td>
</tr>
<tr>
<td>See Also</td>
<td><a href="#">VFSGetDefaultDirectory</a>, <a href="#">VFSRegisterDefaultDirectory</a></td>
</tr>
</tbody>
</table>

## VFSVolumeEnumerate

**Purpose**
Enumerate the mounted volumes.

**Declared In**
VfsMgr.h

**Prototype**
```c
Err VFSVolumeEnumerate (UInt16 *volRefNumP, UInt32 *volIteratorP)
```

**Parameters**

- `<- volRefNumP` Pointer to the reference number for the volume represented by the current enumeration, or `vfsInvalidVolRef` if there are no more volumes to be enumerated or an error occurred.

- `<> volIteratorP` Pointer to a variable that holds the index of the current enumeration. Set the variable to `vfsIteratorStart` prior to the first iteration. Each call to `VFSVolumeEnumerate` updates the variable to the index of the next volume. When the last volume is reached, the variable pointed to by `volIteratorP` is set to `vfsIteratorStop`.

**Result**
Returns one of the following error codes:

- `errNone` No error
- `expErrEnumerationEmpty` There are no volumes to enumerate.
sysErrParamErr The value pointed to by volIteratorP is not valid. This error is also returned when volIteratorP is vfsIteratorStop.

Comments This function returns a pointer to the volume reference number in the volRefNumP parameter. In order to traverse all volumes you must make repeated calls to VFSVolumeEnumerate inside a loop. Before the first call to VFSVolumeEnumerate, the variable pointed to by volIteratorP should be initialized to vfsIteratorStart. Each iteration then increments volIteratorP to the next entry after updating volRefNumP. When the last volume is reached, *volIteratorP is set to vfsIteratorStop. If there are no volumes to enumerate, VFSVolumeEnumerate returns expErrEnumerationEmpty when first called.

Example Below is an example of how to use VFSVolumeEnumerate.

```c
UInt16 volRefNum;
UInt32 volIterator = vfsIteratorStart;

while (volIterator != vfsIteratorStop) {
    err = VFSVolumeEnumerate(&volRefNum, &volIterator);
    if (err == errNone) {
        // Do something with the volRefNum
    } else {
        // handle error... possibly by
        // breaking out of the loop
    }
}
```

Compatibility Implemented only if the VFS Manager Feature Set is present.
VFSVolumeFormat

**Purpose**  Format and mount the volume installed in a given slot.

**Declared In**  VfsMgr.h

**Prototype**  Err VFSVolumeFormat (UInt8 flags, 
                      UInt16 fsLibRefNum, 
                      VFSAnyMountParamPtr vfsMountParamP)

**Parameters**

-> flags  Flags that control how the volume should be formatted. Currently, the only flag not reserved is vfsMountFlagsUseThisFileSystem. Pass this flag to cause the volume to be formatted using the file system specified by fsLibRefNum. Pass zero (0) to have the VFS Manager attempt to format the volume using a file system appropriate to the slot.

-> fsLibRefNum  Reference number of the file system library for which the volume should be formatted. This number is obtained through a call to SysLibFind with the name of the library you want to use. If the flags field is not set to vfsMountFlagsUseThisFileSystem, this parameter is ignored.
<-vfsMountParamP
Parameters to be used when formatting the volume and when mounting the volume after it has been formatted. Supply a pointer to either a VFSSlotMountParamType or a VFSPOSEMountParamType structure. Note that you'll need to cast your structure pointer to a VFSAnyMountParamPtr. Set the mountClass field to the appropriate value: if you are mounting to an Expansion Manager slot, set mountClass to VFSMountClass_SlotDriver and initialize slotLibRefNum and slotRefNum to the appropriate values. See the descriptions of VFSAnyMountParamType, VFSSlotMountParamType, and VFSPOSEMountParamType for information on the fields that make up these data structures.

Result
Returns one of the following error codes:

errNone No error.
expErrNotEnoughPower There is insufficient battery power to format and/or mount a volume.
expErrNotOpen The file system library necessary for this call has not been installed or has not been opened.
vfsErrNoFileSystem The VFS Manager cannot find an appropriate file system to handle the request.

Comments
The slot driver currently only supports one volume per slot. If the volume is successfully formatted and mounted, the reference number of the mounted volume is returned in vfsMountParamP->volRefNum. If the format is unsuccessful or cancelled, vfsMountParamP->volRefNum is set to vfsInvalidVolRef.

If vfsMountFlagsUseThisFileSystem is passed as a flag, VFSVolumeFormat attempts to format the volume using the file system library specified by fsLibRefNum. Typically the flag
Virtual File System Manager

VFS Manager Functions

parameter is not set. In this case VFSVolumeFormat tries to find a compatible library to format the volume, as follows:

1. Check to see if the default file system library feature is set. If it is, and if that file system is installed, it is used to format the volume. You can set the default file system using PtrSet; supply sysFileCVFSMgr for the feature creator, and vfsPtrIDDefaultFS for the feature number.

2. Check to see if any of the installed file systems are natively supported for the slot on which the VFS Manager is trying to format. If one of them is, it is used to format the volume.

3. If none of the installed file systems can perform the format using the slot’s native type, a dialog displays warning the user that their media may become incompatible with other devices if they continue with the format. The user may continue or cancel the format. If the user chooses to continue, VFSVolumeFormat formats the volume using the first file system library that was installed.

When calling VFSVolumeFormat, the volume can either be mounted or unmounted. The underlying file system library call requires the volume to be unmounted. VFSVolumeFormat checks to see if the volume is currently mounted and unmounts it, if necessary, using VFSVolumeUnmount before making the file system call. If the file system successfully formats the volume, VFSVolumeFormat mounts it and posts a sysNotifyVolumeMountedEvent notification.

**Example**

The following code excerpt formats a volume on an Expansion Manager slot using a compatible file system.

```c
VFSSlotMountParamType slotParam;
UInt32 slotIterator = expIteratorStart;

slotParam.vfsMountParamP.mountClass = VFSMountClass_SlotDriver;
err = ExpSlotEnumerate(&slotParam.slotRefNum, &slotIterator);
err = ExpSlotLibFind(slotParam.slotRefNum, &slotParam.slotLibRefNum);
err = VFSVolumeFormat(NULL, NULL, (VFSAnyMountParamPtr) & slotParam);
```
Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSVolumeMount

VFSVolumeGetLabel

Purpose Determine the volume label for a particular volume.

Declared In VfsMgr.h

Prototype Err VFSVolumeGetLabel(UInt16 volRefNum, Char *labelP, UInt16 bufLen)

Parameters

- volRefNum Volume reference number returned from VFSVolumeEnumerate.
- labelP Pointer to a character buffer into which the volume name is placed.
- bufLen Length, in bytes, of the labelP buffer.

Result Returns one of the following error codes:

- errNone No error.
- expErrNotOpen The file system library necessary for this call has not been installed or has not been opened.
- vfsErrNoFileSystem The VFS Manager cannot find an appropriate file system to handle the request.
- vfsErrVolumeBadRef The specified volume has not been mounted.
- vfsErrBufferOverflow The value specified in bufLen is not big enough to receive the full volume label.
- vfsErrNameShortened There was an error reading the full volume name. A shortened version is being returned.
Volume reference numbers can change each time you mount a given volume. To keep track of a particular volume, save the volume’s label rather than its reference number. Volume labels can be up to 255 characters long. They can contain any normal character, including spaces and lower case characters, in any character set as well as the following special characters: $ % ' - _ @ ~ ` ! ( ) ^ # & + , ; = [ ].

Implemented only if the VFS Manager Feature Set is present.

Get information about the specified volume.

VfsMgr.h

Err VFSVolumeInfo(UInt16 volRefNum, VolumeInfoType *volInfoP)

Volume reference number returned from VFSVolumeEnumerate.

Pointer to the structure that receives the volume information for the specified volume. See VolumeInfoType for more information on the fields in this data structure.

Returns one of the following error codes:

timeout: No error.

The file system library necessary for this call has not been installed or has not been opened.

The VFS Manager cannot find an appropriate file system to handle the request.
vfsErrVolumeBadRef
The specified volume reference number is invalid.

Compatibility
Implemented only if the VFS Manager Feature Set is present.

See Also
VFSVolumeGetLabel, VFSVolumeSize

VFSVolumeMount

Purpose
Mount the card’s volume on the specified slot.

Declared In
VfsMgr.h

Prototype
Err VFSVolumeMount(UInt8 flags,
UInt16 fsLibRefNum, 
VFSAnyMountParamPtr vfsMountParamP)

Parameters
- -> flags Flags that control how the volume should be mounted. Currently, the only flag not reserved is vfsMountFlagsUseThisFileSystem. Pass this flag to cause the volume to be mounted using the file system specified by fsLibRefNum. Pass zero (0) to have the VFS Manager attempt to mount the volume using a file system appropriate for the slot.

- -> fsLibRefNum Reference number of the file system library for which the volume should be mounted. This number is obtained through a call to SysLibFind with the name of the library you want to use. If the flags field is not set to vfsMountFlagsUseThisFileSystem, this parameter is ignored.
Virtual File System Manager
VFS Manager Functions

->vfsMountParamP
Parameters to be used when mounting the volume after it has been formatted. Supply a pointer to either a VFSSlotMountParamType or a VFSPOSEMountParamType structure. Note that you'll need to cast your structure pointer to a VFSAnyMountParamPtr. Set the mountClass field to the appropriate value: if you are mounting to an Expansion Manager slot, set mountClass to VFSMountClass_SlotDriver and initialize slotLibRefNum and slotRefNum to the appropriate values. See the descriptions of VFSAnyMountParamType, VFSSlotMountParamType, and VFSPOSEMountParamType for information on the fields that make up these data structures.

Result
Returns one of the following error codes:

errNone No error.
expErrNotEnoughPower There is insufficient battery power to mount a volume.
expErrNotOpen The file system library necessary for this call has not been installed or has not been opened.
sysErrParamErr vfsMountParamP was initialized to NULL.
vfsErrNoFileSystem The VFS Manager cannot find an appropriate file system to handle the request.
vfsErrVolumeStillMounted The volume is already mounted with a different file system than was specified in fsLibRefNum.

Comments
The slot driver only supports one volume per slot. The reference number of the mounted volume is returned in vfsMountParamP->volRefNum. If vfsMountFlagsUseThisFileSystem is passed
as a flag, `VFSVolumeMount` attempts to mount the volume using the file system library specified by `fsLibRefNum`. Otherwise `VFSVolumeMount` tries to find a file system library which is able to mount the volume. If none of the installed file system libraries is able to mount the volume, `VFSVolumeMount` attempts to re-format the volume (using `VFSVolumeFormat` and then mount it. If `VFSVolumeMount` manages to successfully mount the volume, it ends by posting a `sysNotifyVolumeMountedEvent` notification.

After `VFSVolumeMount` successfully mounts a volume, it broadcasts `sysNotifyVolumeMountedEvent`. The VFS Manager, upon being notified of this event, searches the newly-mounted volume for `/PALM/start.prc`. If `start.prc` is found in the `/PALM` directory, the VFS Manager copies it to main memory and launches it. If `start.prc` is not found, the VFS Manager switches to the Launcher instead. This behavior can be overridden; see Card Insertion and Removal in Chapter 7, “Expansion,” Palm OS Programmer’s Companion, vol. I.

When `VFSVolumeMount` is called, if the volume is already mounted with a different file system than was specified in `fsLibRefNum`, a `vfsErrVolumeStillMounted` error is returned. If the volume is already mounted with the same file system that is specified in `fsLibRefNum`, or if `vfsMountFlagsUseThisFileSystem` is not set, `VFSVolumeMount` returns `errNone` and sets `volRefNumP` to the reference number of the currently mounted volume.

**Example**

The following code excerpt mounts a volume on an Expansion Manager slot using a compatible file system.

```c
VFSSlotMountParamType slotParam;
UInt32 slotIterator = expIteratorStart;

slotParam.vfsMountParamP.mountClass = VFSMountClass_SlotDriver;
err = ExpSlotEnumerate(&slotParam.slotRefNum, &slotIterator);
err = ExpSlotLibFind(slotParam.slotRefNum, &slotParam.slotLibRefNum);

err = VFSVolumeMount(NULL, NULL, (VFSAnyMountParamPtr) & slotParam);
```
**Virtual File System Manager**

**VFS Manager Functions**

**Compatibility**
Implemented only if the VFS Manager Feature Set is present.

**See Also**
VFSVolumeFormat, VFSVolumeUnmount

**VFSVolumeSetLabel**

**Purpose**
Changes the volume label for a mounted volume.

**Declared In**
VfsMgr.h

**Prototype**
Err VFSVolumeSetLabel(UInt16 volRefNum, const Char *labelP)

**Parameters**
- `volRefNum` Volume reference number returned from VFSVolumeEnumerate.
- `labelP` Pointer to the label to be applied to the specified volume. This string must be null-terminated.

**Result**
Returns one of the following error codes:
- `errNone` No error
- `expErrNotOpen` The file system library necessary for this call has not been installed or has not been opened.
- `vfsErrBadName` The supplied label is invalid.
- `vfsErrNameShortened` Indicates that the label name was too long. A shortened version of the label name was used instead.
- `vfsErrVolumeBadRef` The specified volume has not been mounted.

**Comments**
Volume labels can be up to 255 characters long. They can contain any normal character, including spaces and lower case characters, in any character set as well as the following special characters: $ % ´ - _ @ ~ ` ! ( ) ^ # & + ; = [ ]. See Naming Volumes in Chapter 7.
NOTE: Most clients should not need to call this function. This function may create or delete a file in the root directory, which would invalidate any current calls to `VFSDirEntryEnumerate`.

**Compatibility**
Implemented only if the VFS Manager Feature Set is present.

**See Also**
VFSVolumeGetLabel

## VFSVolumeSize

**Purpose**
Determine the total amount of space on a volume, as well as the amount that is currently being used.

**Declared In**
VfsMgr.h

**Prototype**
```c
Err VFSVolumeSize(UInt16 volRefNum, UInt32 *volumeUsedP, UInt32 *volumeTotalP)
```

**Parameters**
- `-> volRefNum` Volume reference number returned from `VFSVolumeEnumerate`.
- `<- volumeUsedP` Pointer to a variable that receives the amount of space, in bytes, in use on the volume.
- `<- volumeTotalP` Pointer to a variable that receives the total amount of space on the volume, in bytes.

**Result**
Returns one of the following error codes:
- `errNone` No error.
- `expErrNotOpen` The file system library necessary for this call has not been installed or has not been opened.
vfsErrNoFileSystem
   The VFS Manager cannot find an appropriate
   file system to handle the request.

vfsErrVolumeBadRef
   The specified volume has not been mounted.

Compatibility
   Implemented only if the VFS Manager Feature Set is present.

See Also  VFSVolumeInfo

VFSVolumeUnmount

Purpose  Unmount the given volume.

Declared In  VfsMgr.h

Prototype  Err VFSVolumeUnmount(UInt16 volRefNum)

Parameters
   -> volRefNum
      Volume reference number returned from
      VFSVolumeEnumerate.

Result
   Returns one of the following error codes:
   errNone  No error.
   expErrNotOpen  The file system library necessary for this call
      has not been installed or has not been opened.
   vfsErrNoFileSystem  The VFS Manager cannot find an appropriate
      file system to handle the request.
   vfsErrVolumeBadRef  The specified volume has not been mounted.

Comments
   This function closes any opened files and posts a
   sysNotifyVolumeUnmountedEvent notification once the file
   system is successfully unmounted.
Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSVolumeMount

Application-Defined Functions

VFSExportProcPtr

Purpose User-defined callback function supplied to VFSExportDatabaseToFileCustom that tracks the progress of the export.

Declared In VfsMgr.h

Prototype typedef Err (*VFSExportProcPtr)(UInt32 totalBytes, UInt32 offset, void *userDataP)

Parameters

- totalBytes: The total number of bytes being exported.
- offset: Undefined.
- userDataP: Pointer to any application-specific data passed to the callback function. This pointer may be NULL if your callback doesn’t need any such data.

Result Your progress tracker should allow the user to abort the export. Return errNone if the export should continue, or any other value to abort the export process. If you return a value other than errNone, that value will be returned by VFSExportDatabaseToFileCustom.

Comments See the Progress Dialogs section in the Palm OS Programmer’s Companion, vol. I for more information on writing a progress tracker.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSImportProcPtr
VFSImportProcPtr

**Purpose**
User-defined callback function supplied to `VFSImportDatabaseFromFileCustom` that tracks the progress of the import.

**Declared In**
`VfsMgr.h`

**Prototype**
```c
typedef Err (*VFSImportProcPtr)
(UInt32 totalBytes, UInt32 offset, void *userDataP)
```

**Parameters**
- `totalBytes` The total number of bytes being imported.
- `offset` The number of bytes that have already been imported. This value, along with the total number of bytes being imported, allows you to inform the user how far along the import is.
- `userDataP` Pointer to NY application-specific data passed to the callback function. This pointer may be `NULL` if your callback doesn’t need any such data.

**Result**
Your progress tracker should allow the user to abort the import. Return `errNone` if the import should continue, or any other value to abort the import process. If you return a value other than `errNone`, that value will be returned by `VFSImportDatabaseFromFileCustom`.

**Comments**
See the Progress Dialogs section in the Palm OS Programmer’s Companion, vol. I for more information on writing a progress tracker.

**Compatibility**
Implemented only if the VFS Manager Feature Set is present.

**See Also**
`VFSExportProcPtr`
Windows

This chapter provides information about windows by discussing these topics:

- Window Data Structures
- Window Constants
- Window Functions

No resources are associated with window objects.

The header file Window.h declares the API that this chapter describes. For more information on windows, see the section “Text” in the Palm OS Programmer’s Companion, vol. I.

Window Data Structures

**CustomPatternType**

The CustomPatternType type holds an 8-by-8 bit pattern that is one bit deep. Each byte specifies a row of the pattern. When drawing, a pattern is tiled to fill a specified region. This pattern is used by WinFillLine and WinFillRectangle.

The PatternType specifies the name of the current pattern.

```c
typedef UInt8 CustomPatternType [8];
```

**Compatibility**

In pre-3.5 systems, the CustomPatternType is an array of 4 16-bit words. Note the size of this data type has not changed.

**DrawStateType**

The DrawStateType structure defines the current drawing state, which is the Palm OS® implementation of a pen. This drawing state is saved with WinPushDrawState and restored with WinPopDrawState.
WARNING!    PalmSource, Inc. does not support or provide backward compatibility for the DrawStateType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

typedef struct DrawStateType {
    WinDrawOperation transferMode;
    PatternType pattern;
    UnderlineModeType underlineMode;
    FontID fontId;
    FontPtr font;
    CustomPatternType patternData;
    IndexedColorType foreColor;
    IndexedColorType backColor;
    IndexedColorType textColor;
    UInt8 reserved;
    RGBColorType foreColorRGB;
    RGBColorType backColorRGB;
    RGBColorType textColorRGB;
    UInt16 coordinateSystem;
    UInt16 reserved2;
    Fixed16 scale;
    Fixed16 ntvToActiveScale;
    Fixed16 stdToActiveScale;
    Fixed16 activeToStdScale;
} DrawStateType;
Field Descriptions

**transferMode**  
The current transfer mode for color drawing. See [WinDrawOperation](#). Use [WinSetDrawMode](#) to set this value.

**pattern**  
The name of the current pattern. See [PatternType](#). If set to customPattern, the patternData field contains the actual pattern. Use [WinGetPatternType](#) and [WinSetPatternType](#) to retrieve and set this value.

**underlineMode**  
The current underline mode. See [UnderlineModeType](#). Use [WinSetUnderlineMode](#) to set this value.

**fontId**  
The ID of the current font. Use FntSetFont to set this value.

**font**  
A pointer to the current font. Use FntSetFont to set this value.

**patternData**  
The current pattern being used by the WinFill functions if pattern is customPattern. See [CustomPatternType](#). Use [WinGetPattern](#) and [WinSetPattern](#) to retrieve and set this value.

**foreColor**  
Index of the current color used for the foreground. Use WinSetForeColor to set this value. This field is only valid for indexed color bitmaps.

**backColor**  
Index of the current color used for the background. Use WinSetBackColor to set this value. This field is only valid for indexed color bitmaps.

**textColor**  
Index of the current color used for text. Use WinSetTextColor to set this value. This field is only valid for indexed color bitmaps.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reserved</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>foreColorRGB</td>
<td>RGB value of the current color used for the foreground. Use <code>WinSetForeColorRGB</code> to set this value. This field is only valid for Palm OS 4.0 or later, and only valid for direct color bitmaps.</td>
</tr>
<tr>
<td>backColorRGB</td>
<td>RGB value of the current color used for the background. Use <code>WinSetBackColorRGB</code> to set this value. This field is only valid for Palm OS 4.0 or later, and only valid for direct color bitmaps.</td>
</tr>
<tr>
<td>textColorRGB</td>
<td>RGB value of the current color used for text. Use <code>WinSetTextColorRGB</code> to set this value. This field is only valid for Palm OS 4.0 or later, and only valid for direct color bitmaps.</td>
</tr>
<tr>
<td>coordinateSystem</td>
<td>Active coordinate system.</td>
</tr>
<tr>
<td>reserved2</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>scale</td>
<td>A fixed point value used to convert from the draw window’s active coordinate system to native coordinates. This field is defined only if the High-Density Display Feature Set is present.</td>
</tr>
<tr>
<td>ntvToActiveScale</td>
<td>A fixed point value used to convert from the native coordinate system to the draw window’s active coordinate system. This field is defined only if the High-Density Display Feature Set is present.</td>
</tr>
</tbody>
</table>
stdToActiveScale: A fixed point value used to convert from the standard coordinate system to the draw window’s active coordinate system. This field is used internally to convert font metrics, which are stored as standard coordinates. This field is defined only if the High-Density Display Feature Set is present.

activeToStdScale: The inverse of stdToActive; the active-to-standard scaling factor. This field is defined only if the High-Density Display Feature Set is present.

Compatibility: This type is implemented only if 3.5 New Feature Set is present. The scale, ntvToActiveScale, stdToActiveScale, and activeToStdScale fields are defined only if the High-Density Display Feature Set is present.

FrameBitsType

The FrameBitsType structure specifies attributes of a window’s frame.

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the FrameBitsType bit field. Never access its bit field members directly, or your code may break in future versions. Use the information below for debugging purposes only.

typedef union FrameBitsType {
    struct {
        UInt16 cornerDiam : 8;
        UInt16 reserved_3 : 3;
        UInt16 threeD : 1;
        UInt16 shadowWidth : 2;
        UInt16 width : 2;
    } bits;
    UInt16 word;
} FrameBitsType;
Field Descriptions

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cornerDiam</td>
<td>Corner radius of frame; maximum is 38.</td>
</tr>
<tr>
<td>reserved_3</td>
<td>Reserved.</td>
</tr>
<tr>
<td>threeD</td>
<td>Set this bit to draw a 3D button. This feature is not currently supported.</td>
</tr>
<tr>
<td>shadowWidth</td>
<td>Width of shadow.</td>
</tr>
<tr>
<td>width</td>
<td>Frame width.</td>
</tr>
<tr>
<td>word</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

### FrameType

The `FrameType` type specifies a window frame style.

```c
typedef UInt16 FrameType;
```

The `FrameType` can be set to one of the defined frame types shown in the table below, or a custom frame type as defined by a `FrameBitsType` structure.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noFrame</td>
<td>0</td>
<td>No frame</td>
</tr>
<tr>
<td>simpleFrame</td>
<td>1</td>
<td>Plain rectangular frame</td>
</tr>
<tr>
<td>rectangleFrame</td>
<td>1</td>
<td>Plain rectangular frame</td>
</tr>
<tr>
<td>simple3DFrame</td>
<td>0x0012</td>
<td>3D frame with width of 2. This frame type is not supported.</td>
</tr>
<tr>
<td>roundFrame</td>
<td>0x0401</td>
<td>Round frame with width of 1.</td>
</tr>
<tr>
<td>boldRoundFrame</td>
<td>0x0702</td>
<td>Round frame with width of 2.</td>
</tr>
<tr>
<td>popupFrame</td>
<td>0x0205</td>
<td>Popup frame style with slight corner roundness, width of 1 and shadow of 1.</td>
</tr>
<tr>
<td>dialogFrame</td>
<td>0x0302</td>
<td>Dialog frame style with slight corner roundness and width of 2.</td>
</tr>
<tr>
<td>menuFrame</td>
<td>popupFrame</td>
<td>Same as popupFrame.</td>
</tr>
</tbody>
</table>

1152  *Palm OS Programmer’s API Reference*
IndexedColorType
The IndexedColorType type is used to specify a color by its index value; that is, by its location in a color table. Color tables are defined by the ColorTableType structure, which is declared in Bitmap.h. The IndexedColorType can hold a 1, 2, 4, or 8-bit index.

```c
typedef UInt8 IndexedColorType;
```

Compatibility
This type is implemented only if 3.5 New Feature Set is present.

PatternType
The PatternType enumerated type specifies a pattern for drawing. This type is returned by WinGetPatternType and is used as a parameter to the WinSetPatternType function.

```c
typedef enum {
    blackPattern,
    whitePattern,
    grayPattern,
    customPattern,
    lightGrayPattern,
    darkGrayPattern
} PatternType;
```

Value Descriptions
- **blackPattern** Pattern with all bits on.
- **whitePattern** Pattern with all bits off.
- **grayPattern** Pattern with alternating on and off bits.
- **customPattern** Custom pattern specified by CustomPatternType.
These patterns all operate with current foreground and background color instead of black and white. In effect, `blackPattern` is only black if the current foreground color is black. `whitePattern` uses the current background color. `grayPattern` and `customPattern` use a combination of background and foreground colors.

Patterns are expanded to the destination bit depth by the blitter when drawing patterned lines and filled rectangles.

The three standard gray patterns—`grayPattern`, `lightGrayPattern`, and `darkGrayPattern`—are always drawn by the blitter using the screen density to improve the appearance of gray fills. Custom patterns, however, are stretched as appropriate by the blitter based on the destination density.

**Compatibility**

The `lightGrayPattern` and `darkGrayPattern` values are defined only if the [High-Density Display Feature Set](#) is present.

**UnderlineModeType**

The `UnderlineModeType` enumerated type specifies possible values for the underline mode stored in `DrawStateType`.
typedef enum { noUnderline, grayUnderline, solidUnderline, colorUnderline } UnderlineModeType;

**Value Descriptions**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noUnderline</td>
<td>No underline.</td>
</tr>
<tr>
<td>grayUnderline</td>
<td>Underline is drawn using a dotted line in the current foreground color.</td>
</tr>
<tr>
<td>solidUnderline</td>
<td>Underline is drawn using a solid line in the foreground color.</td>
</tr>
<tr>
<td>colorUnderline</td>
<td>Underline is drawn using a solid line in the foreground color.</td>
</tr>
</tbody>
</table>

**Compatibility**
The solidUnderline and colorUnderline options are only available in Palm OS 3.1 and higher.

**WindowFlagsType**
The WindowFlagsType specifies different window attributes.

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the WindowFlagsType bit field. Access it only through the functions described below. Never access its bit field members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```c
typedef struct WindowFlagsType {
    UInt16 format:1;
    UInt16 offscreen:1;
    UInt16 modal:1;
    UInt16 focusable:1;
    UInt16 enabled:1;
    UInt16 visible:1;
    UInt16 dialog:1;
    UInt16 freeBitmap:1;
```
Field Descriptions

format  If set, use the genericFormat. If 0, use screenFormat. Screen format is the native format of the video system; windows in this format can be copied to the display faster. The generic format is device-independent. A window cannot be enabled (that is, accept pen input) unless it uses screen format.

offscreen  If set, the window is offscreen. If 0, the window is onscreen.

modal  If set, the window is modal. If 0, the window is not modal. You set this value when you create the window. This value is returned by WinModal.

focusable  If set, the window can accept the focus. If 0, the window does not accept the focus. You set this value when you create the window.

enabled  If set, the window is enabled. If 0, the window is disabled.

visible  If set, the window is visible if it is onscreen. If 0, the window is not visible.

dialog  If set, the window is a form. If 0, the window is not a form. The FrmInitForm function sets this value.

freeBitmap  If set, free the bitmap when the window is freed. If 0, retain the bitmap after the window is freed.

reserved  Reserved for future use. Must be 0.

Compatibility  In OS versions previous to 3.5, the freeBitmap flag was not present. Instead, a compressed flag was present, where 0 specified uncompressed and 1 specified compressed. This compressed flag is now part of the BitmapType.
WindowFormatType

Use this enumeration to specify the window format when creating an offscreen window with the WinCreateOffscreenWindow function.

```c
typedef enum {
    screenFormat = 0,
    genericFormat,
    nativeFormat
} WindowFormatType;
```

**Field Values**

**screenFormat** The window’s bitmap is allocated using the hardware screen’s depth, but for backward compatibility the bitmap associated with the offscreen window is always low density, and the window always uses a coordinate system that directly maps offscreen pixels to coordinates.

**genericFormat** Like `screenFormat`, except that `genericFormat` offscreen windows do not accept pen input.

**nativeFormat** Reflects the actual hardware screen format in all ways, including screen depth, density, and pixel format. Applications must always use the graphic API when drawing to a `nativeFormat` offscreen window: directly accessing offscreen pixels will produce undefined results. When using this format, the width and height arguments must be specified using the active coordinate system. Like `genericFormat`, `nativeFormat` offscreen windows do not accept pen input.

**Compatibility** WindowFormatType is defined only if the High-Density Display Feature Set is present.

**WindowType**

The WindowType structure represents a window.
**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `WindowType` structure. Access it only through the functions described below. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```c
typedef struct WindowType {
    Coord displayWidthV20;
    Coord displayHeightV20;
    void  *displayAddrV20;
    WindowFlagsType windowFlags;
    RectangleType windowBounds;
    AbsRectType clippingBounds;
    BitmapPtr bitmapP;
    FrameBitsType frameType;
    DrawStateType *drawStateP;
    struct WindowType *nextWindow;
} WindowType;
```

**Field Descriptions**

- `displayWidthV20`  Width of the window in pre OS 3.5 devices. In OS 3.5, use `WinGetBounds` to return the window width.

- `displayHeightV20` Height of the window in pre OS 3.5 devices. In OS 3.5, use `WinGetBounds` to return the window height.

- `displayAddrV20`  Pointer to the window display memory buffer in pre OS 3.5 devices. In OS 3.5 or later, call `WinGetBitmap` and then `BmpGetBits` to obtain the window’s memory buffer.

**WARNING!** Writing directly to screen memory will not be supported in all devices.

- `windowFlags`  Window attributes (see `WindowFlagsType`).
Compatibility

In OS versions previous to 3.5, this structure is slightly different. Specifically, the bitmapP field is instead a viewOrigin field of type PointType and specified the window origin point on the display. The drawStateP was named gstate and was of type GraphicStatePtr. The complete definition is shown below:

```c
typedef struct WinTypeStruct {
    Word                   displayWidth;
    Word                   displayHeight;
    VoidPtr                displayAddr;
    WindowFlagsType        windowFlags;
    RectangleType          windowBounds;
    AbsRectType            clippingBounds;
    PointType              viewOrigin;
    FrameBitsType          frameType;
    GraphicStatePtr        gstate;
    struct WinTypeStruct   *nextWindow;
} WinType;
```
WinDrawOperation

The WinDrawOperation enumerated type specifies the transfer mode for color drawing. This type is used as a parameter to the WinCopyRectangle and WinSetDrawMode functions.

```c
typedef enum {winPaint, winErase, winMask, winInvert, winOverlay, winPaintInverse, winSwap} WinDrawOperation;
```

Value Descriptions

- **winPaint**: Write color-matched source pixels to the destination. If a bitmap’s hasTransparency flag is set, winPaint behaves like winOverlay instead.

- **winErase**: Write backColor if the source pixel is transparent.

- **winMask**: Write backColor if the source pixel is not transparent.

- **winInvert**: Bitwise XOR the color-matched source pixel onto the destination. This mode does not honor the transparent color in any way.

- **winOverlay**: Write color-matched source pixel to the destination if the source pixel is not transparent. Transparent pixels are skipped. For a 1-bit display, the “off” bits are considered to be the transparent color. Note that this definition of winOverlay is new for Palm OS 5: in Palm OS 4.x, the destination is set (only) where the source pixels are “on.”
Note that 2-bit, 4-bit, and 8-bit source bitmaps that don’t have a color table inherit the system default color table for their given depth. 1-bit sources (bitmap, text, and patterns) that don’t have a color table are given a color table where entry 0 is the backColor and entry 1 is the foreColor (textColor for text).

winSwap is not a color invert operation, although a pair of winSwap operations will restore the original graphics data. This mode is used by the OS to select and deselect areas of the screen. It changes destination pixels matching the foreground color to the background color, and changes destination pixels matching the background color to the foreground color. It is a mode available for rectangles, lines, and pixels, but not text or bitmaps. This mode ignores the current pattern. **The Transparent Color**

As of Palm OS 4.0, bitmaps have a hasTransparency flag and may designate a transparent color. These concepts are augmented somewhat in Palm OS 5 to make the transfer modes more consistent:

- Bitmaps that don’t specify any transparent color (text, patterns, and version 0 bitmaps) are assumed to have a transparent color of index 0 and the hasTransparency bit is assumed to be false.

- When the hasTransparency flag is set and the transfer mode is winPaint, only the non-transparent pixels are copied to the destination. With text and patterns, the Palm OS assumes that the “off” bits are the ones designated as transparent and acts as if the hasTransparency flag is always false. This assumption retains backwards compatibility and unifies the use of transparency across all source data.
Windows
Window Data Structures

Compatibility
This type is implemented only if 3.5 New Feature Set is present. In earlier releases, this type is named ScrOperation and its values begin with the prefix scr rather than win. WinDrawOperation is fully compatible with ScrOperation. Transparency is only available if 4.0 New Feature Set is present, and as mentioned in “winSwap is not a color invert operation, although a pair of winSwap operations will restore the original graphics data. This mode is used by the OS to select and deselect areas of the screen. It changes destination pixels matching the foreground color to the background color, and changes destination pixels matching the background color to the foreground color. It is a mode available for rectangles, lines, and pixels, but not text or bitmaps. This mode ignores the current pattern. The Transparent Color” the behavior of the window drawing modes—winOverlay in particular—changes slightly if 5.0 New Feature Set is present.

WinHandle
The WinHandle type is a pointer to a WindowType structure. Note that this may change.

typedef WindowType *WinHandle;

WinLineType
The WinLineType structure defines a line.

typedef struct WinLineType {
    Coord x1;
    Coord y1;
    Coord x2;
    Coord y2;
} WinLineType;

Field Descriptions
x1 X coordinate of the first endpoint of the line.
y1 Y coordinate of the first endpoint of the line.
x2 X coordinate of the second endpoint of the line.
y2 Y coordinate of the second endpoint of the line.
Compatibility

This type is implemented only if 3.5 New Feature Set is present.

**WinPtr**

The WinPtr type is a pointer to a WindowType structure.

typedef WindowType *WinPtr;

**Window Constants**

If the 5.0 New Feature Set or the High-Density Display Feature Set are present, the following constants are defined:

- **Window Coordinate System Constants**

### New Window Coordinate System Constants

These constants specify the coordinate system to be used when drawing within a given window:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kCoordinatesNative</td>
<td>0</td>
<td>Use the bitmap’s native coordinate system; this enables a 1-to-1 correspondence between coordinates and pixels.</td>
</tr>
<tr>
<td>kCoordinatesStandard</td>
<td>72</td>
<td>The coordinate system used by most handhelds running Palm OS 4.0 and earlier. On a single-density handheld, there is one screen pixel per standard coordinate. On a high-density screen, there is more than one screen pixel per standard coordinate.</td>
</tr>
<tr>
<td>kCoordinatesOneAndAHalf</td>
<td>108</td>
<td>One and a half times the standard coordinate system.</td>
</tr>
<tr>
<td>kCoordinatesDouble</td>
<td>144</td>
<td>Twice the standard coordinate system.</td>
</tr>
<tr>
<td>kCoordinatesTriple</td>
<td>216</td>
<td>Three times the standard coordinate system.</td>
</tr>
<tr>
<td>kCoordinatesQuadruple</td>
<td>288</td>
<td>Four times the standard coordinate system.</td>
</tr>
</tbody>
</table>
IMPORTANT: Not all coordinate systems listed in the above table are supported in this version of the High-Density Display Feature Set. For Palm OS 5, only kCoordinatesNative, kCoordinatesStandard, and kCoordinatesDouble are supported.

Pass one of these constants as an argument to WinSetCoordinateSystem. These values are returned by WinGetCoordinateSystem.

Compatibility Defined only if the High-Density Display Feature Set is present.

Window Functions

WinClipRectangle

Purpose Shrinking the rectangle to make it fit within the clipping region of the current draw window.

Declared In Window.h

Prototype void WinClipRectangle (RectangleType *rP)

Parameters <-> rP Pointer to a structure holding the rectangle to clip. The rectangle returned is the intersection of the rectangle passed and the clipping bounds of the draw window.

Result Returns nothing.

Comments This function does not change the clipping rectangle of the window. To modify the window’s clipping rectangle, use the WinSetClip and WinResetClip functions.
The draw window is the window to which all drawing functions send their output. It is returned by \texttt{WinGetDrawWindow}.

\textbf{See Also} \texttt{WinCopyRectangle}, \texttt{WinDrawRectangle}, \texttt{WinEraseRectangle}, \texttt{WinGetClip}

\section*{WinCopyRectangle}

\textbf{Purpose} Copy a rectangular region from one place to another (either between windows or within a single window).

\textbf{Declared In} Window.h

\textbf{Prototype} \begin{verbatim} void WinCopyRectangle (WinHandle srcWin, WinHandle dstWin, const RectangleType *srcRect, Coord destX, Coord destY, WinDrawOperation mode) \end{verbatim}

\textbf{Parameters}  

-> \texttt{srcWin} \hspace{1cm} Window from which the rectangle is copied. If NULL, use the draw window.  
-> \texttt{dstWin} \hspace{1cm} Window to which the rectangle is copied. If NULL, use the draw window.  
-> \texttt{srcRect} \hspace{1cm} Bounds of the region to copy.  
-> \texttt{destX} \hspace{1cm} Top bound of the rectangle in destination window.  
-> \texttt{destY} \hspace{1cm} Left bound of the rectangle in destination window.  
-> \texttt{mode} \hspace{1cm} The method of transfer from the source to the destination window (see \texttt{WinDrawOperation}).

\textbf{Result} \hspace{1cm} Returns nothing.

\textbf{Comments} Copies the bits of the window inside the rectangle region.  
If the destination bitmap is compressed, the mode parameter must be \texttt{winPaint}, and the destination coordinates must be (0,0). If the width of the destination rectangle is less than 16 pixels or if the
destination coordinates are not (0,0), then this function turns off compression for the destination bitmap. Normally, you do not copy to a compressed bitmap. Instead, you copy to an uncompressed bitmap and compress it afterwards.

**Compatibility**

In OS versions before 3.5, the mode parameter was defined as type ScrOperation. It is defined as type WinDrawOperation only if 3.5 New Feature Set is present. ScrOperation and WinDrawOperation are fully compatible with each other.

In OS versions before 3.5, it was common practice to render a bitmap in an offscreen window and then use WinCopyRectangle to draw it on the screen. In version 3.5 and higher, the preferred method of doing this is to use WinDrawBitmap or WinPaintBitmap.

**See Also**  WinDrawBitmap

---

**WinCreateBitmapWindow**

**Purpose**  Create a new offscreen window.

**Declared In**  Window.h

**Prototype**  
WinHandle WinCreateBitmapWindow
(BitmapType *bitmapP, UInt16 *error)

**Parameters**

- -> bitmapP  Pointer to a bitmap to associate with the window. (See BitmapType.)
- <- error  Pointer to any error this function encounters.

**Result**  Returns the handle of the new window upon success, or NULL if an error occurs. The error parameter contains one of the following:

- errNone  No error.
- sysErrParamErr  The bitmap parameter is invalid. The bitmap must be uncompressed and it must have a valid pixel size (1, 2, 4, or 8). It must not be the screen bitmap.
sysErrNoFreeResource

There is not enough memory to allocate a new window structure.

Comments

Use WinCreateBitmapWindow if you want to draw into a previously created bitmap, such as a bitmap created using BmpCreate.

This function generates a window wrapper for the specified bitmap. The newly created window is offscreen, uses the generic format (for device independence), and is added to the active window list. Use WinSetDrawWindow to make it the draw window, and then use the window drawing functions to modify the bitmap.

When you use this function to create a window and then delete the window with WinDeleteWindow, the bitmap is not freed when the window is freed.

WinCreateOffscreenWindow uses this function to create its offscreen window. If you call WinCreateOffscreenWindow instead of using this function, the bitmap is freed when WinDeleteWindow is called.

The bitmap data will not be blitted properly if the depth of the screen is changed using WinScreenMode and the new window uses a bitmap that does not define the bitmap’s color table. See WinScreenMode for information on how to work around this limitation.

Compatibility

Implemented only if 3.5 New Feature Set is present.

See Also

WinCreateWindow, WinCreateOffscreenWindow
WinCreateOffscreenWindow

**Purpose**
Create a new offscreen window and add it to the window list.

**Declared In**
Window.h

**Prototype**
WinHandle WinCreateOffscreenWindow (Coord width, Coord height, WindowFormatType format, UInt16 *error)

**Parameters**
- `width` Width of the window in pixels. The coordinate system you use for this parameter depends upon the value of `format`.
- `height` Height of the window in pixels. The coordinate system you use for this parameter depends upon the value of `format`.
- `format` One of the window formats defined by `WindowFormatType`.
- `error` Pointer to any error this function encounters.

**Result**
Returns the handle of the new window upon success, or `NULL` if an error occurs. The `error` parameter contains one of the following:
- `errNone` No error.
- `sysErrParamErr` The width or height parameter is `NULL` or the current color table is invalid.
- `sysErrNoFreeResource` There is not enough memory to complete the function.

The debug ROM gives a warning if you try to draw to a bad window address.

**Comments**
Windows created with this routine draw to a memory buffer instead of the display. Use this function for temporary drawing operations such as double-buffering or save-behind operations.

The memory buffer has two formats: screen format and generic format. Screen format is the native format of the video system;
windows in this format can be copied to the display faster. The
generic format is device-independent. A window cannot be enabled
(that is, accept pen input) unless it uses screen format.

This function differs from WinCreateBitmapWindow in the
following ways:

• WinCreateOffscreenWindow creates a new bitmap in the
  same depth as the current screen.
  WinCreateBitmapWindow uses the bitmap you pass in,
  which may or may not be in the same depth as the current
  screen.

• WinCreateOffscreenWindow uses the screen format you
  specify. WinCreateBitmapWindow always uses
genericFormat for the format argument.

• When you delete the window created with
  WinCreateOffscreenWindow, its bitmap is freed along
  with the window. The bitmap used in the
  WinCreateBitmapWindow is not freed when the window is
  freed.

Note that if you aren’t directly accessing the bits of an offscreen
window’s bitmap but are just using the APIs, you can always pass
nativeFormat for the screen format even on pre-Palm OS 5
handhelds and things will work as expected. If you need direct
access to the bits of the offscreen window’s bitmap, however, call
BmpCreate and then call WinCreateBitmapWindow. Because you
created the bitmap, you know its format and thus can safely
manipulate its bits. Calling WinCreateOffscreenWindow with a
format argument of nativeFormat can result in a bitmap with an
unexpected format: the endianness, number of bits per pixel, and so
on would match the screen and therefore be fastest to draw, but
your application wouldn’t be able to manipulate the pixels directly.

The bitmap data will not be blitted properly if the depth of the
screen is changed using WinScreenMode and the new window
uses a bitmap that does not define the bitmap’s color table. See
WinScreenMode for information on how to work around this
limitation.

See Also  WinCreateWindow, WinScreenMode,
WinCreateWindow

**Purpose** Create a new window and add it to the window list.

**Declared In** Window.h

**Prototype**

```
WinHandle WinCreateWindow
( const RectangleType *bounds, FrameType frame,
  Boolean modal, Boolean focusable, UInt16 *error)
```

**Parameters**

- `-> bounds` Display-relative bounds of the window.
- `-> frame` Type of frame around the window (see `FrameType`).
- `-> modal` true if the window is modal.
- `-> focusable` true if the window can be the active window.
- `<- error` Pointer to any error encountered by this function.

**Result** Returns the handle of the new window upon success, or NULL if an error occurs. The `error` parameter contains one of the following:

- `errNone` No error.
- `sysErrNoFreeResource` There is not enough memory to complete the operation.

**Comments** Windows created by this routine draw to the display. See `WinCreateOffscreenWindow` for information on drawing off screen.

You typically don’t call this function directly. Instead, you use `FrmInitForm` to create form windows from a resource. Forms are much more flexible and have better system support. All forms are windows, but not all windows are forms.

The window is created with the bounds and frame type that you specify and uses the bitmap and drawing state of the current draw window. Its clipping region is reset according to the bounds you specify.
All window flags are set to 0 except for the modal and focusable flags, which you pass as a parameter to this function. Specifically, newly created windows are disabled and invisible. You must specifically enable the window before the window can accept input. You can do so with WinSetActiveWindow.

See Also WinDeleteWindow

WinDeleteWindow

Purpose Remove a window from the window list and free the memory used by the window.

Declared In Window.h

Prototype void WinDeleteWindow (WinHandle winHandle, Boolean eraseIt)

Parameters

- > winHandle Handle of window to delete.
- > eraseIt If true, the window is erased before it is deleted. If false, the window is not erased.

Result Returns nothing.

Comments This function frees all memory associated with the window. Windows created using WinCreateOffscreenWindow have their bitmaps freed; windows created using WinCreateWindow or WinCreateBitmapWindow do not.

The eraseIt parameter affects onscreen windows only; offscreen windows are never erased. As a performance optimization, you might set eraseIt to false for an onscreen window if you know that you are going to immediately redraw the area anyway. For example, when the form manager closes a form dialog, it restores the area with the save-behind bits it had stored for that form. For this reason, when the form manager deletes the dialog window, it passes false for eraseIt because the entire area will be redrawn.
WinDisplayToWindowPt

Purpose
Convert a display-relative coordinate to a window-relative coordinate. The coordinate returned is relative to the display window.

Declared In
Window.h

Prototype
void WinDisplayToWindowPt (Coord *extentX, Coord *extentY)

Parameters
<-> extentX Pointer to x coordinate to convert.
<-> extentY Pointer to y coordinate to convert.

Result
Returns nothing.

See Also
WinWindowToDisplayPt

WinDrawBitmap

Purpose
Draw a bitmap at the given coordinates in winPaint mode (see WinDrawOperation for mode details).

Declared In
Window.h

Prototype
void WinDrawBitmap (BitmapPtr bitmapP, Coord x, Coord y)

Parameters
-> bitmapP Pointer to a bitmap.
-> x The x coordinate of the top-left corner.
-> y The y coordinate of the top-left corner.

Result
Returns nothing.

Comments
If the bitmap has multiple depths (is a bitmap family), the closest match less than or equal to the current draw window depth is used.
If such a bitmap does not exist, the bitmap with the closest match greater than the draw window depth is used.

If the bitmap has its own color table, color conversion to the draw window color table will be applied (on OS 3.5 or later). This color conversion is slow and not recommended. Instead of including a color table in the bitmap, consider using WinPalette to change the system color table, draw the bitmap, and then change the system color table back when the bitmap is no longer visible.

This function differs from WinPaintBitmap in that this function always uses winPaint mode (copy mode) as the transfer mode. WinPaintBitmap uses the current drawing state transfer mode.

See Also  WinEraseRectangle

WinDrawChar

Purpose  Draw the specified character in the draw window.

Declared In  Window.h

Prototype  void WinDrawChar (WChar theChar, Coord x, Coord y)

Parameters  -> theChar  The character to draw. This may be either a single-byte character or a multi-byte character.

-> x  x coordinate of the location where the character is to be drawn (left bound).

-> y  y coordinate of the location where the character is to be drawn (top bound).

Result  Returns nothing.

Comments  Before calling this function, call WinSetUnderlineMode and FntSetFont to set the desired underline and font to draw the characters.

This function differs from WinPaintChar in that this function always uses winPaint mode (see WinDrawOperation). This
means the on bits are drawn in the text color, the off bits are in the background color, and underlines are in the foreground color. WinPaintChar uses the current drawing state transfer mode instead of winPaint.

**Compatibility**

Implemented only if 3.1 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call WinGlueDrawChar. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also** WinDrawChars, WinDrawInvertedChars, WinDrawTruncChars, WinEraseChars, WinInvertChars, WinPaintChars

---

**WinDrawChars**

**Purpose**

Draw the specified characters in the draw window.

**Declared In**

Window.h

**Prototype**

```c
void WinDrawChars (const Char *chars, Int16 len, Coord x, Coord y)
```

**Parameters**

- `chars` Pointer to the characters to draw.
- `len` Length in bytes of the characters to draw.
- `x` x coordinate of the first character to draw (left bound).
- `y` y coordinate of the first character to draw (top bound).

**Result**

Returns nothing.

**Comments**

This function is useful for printing non-editable status or warning messages on the screen.

Before calling this function, call WinSetUnderlineMode and FntSetFont to set the desired underline and font to draw the characters.
This function differs from WinPaintChars in that this function always uses winPaint mode (see WinDrawOperation). This means the on bits are drawn in the text color, the off bits are in the background color, and underlines are in the foreground color. WinPaintChar uses the current drawing state transfer mode instead of winPaint.

See Also WinDrawChar, WinDrawInvertedChars, WinDrawTruncChars, WinEraseChars, WinInvertChars, WinPaintChar

WinDrawGrayLine

Purpose Draw a dashed line in the draw window.

Declared In Window.h

Prototype void WinDrawGrayLine (Coord x1, Coord y1, Coord x2, Coord y2)

Parameters

- > x1 x coordinate of line start point.
- > y1 y coordinate of line start point.
- > x2 x coordinate of line endpoint.
- > y2 y coordinate of line endpoint.

Result Returns nothing.

Comments This routine does not draw in the gray color; it draws with alternating foreground and background pixels. That is, it uses the grayPattern pattern type.

See Also WinDrawLine, WinEraseLine, WinFillLine, WinInvertLine, WinPaintLine, WinPaintLines
WinDrawGrayRectangleFrame

**Purpose**
Draw a gray rectangular frame in the draw window.

**Declared In**
Window.h

**Prototype**
```c
void WinDrawGrayRectangleFrame (FrameType frame, const RectangleType *rP)
```

**Parameters**
- `frame` Type of frame to draw (see `FrameType`).
- `rP` Pointer to the rectangle to frame.

**Result**
Returns nothing.

**Comments**
This routine does not draw in the gray color; it draws with alternating foreground and background pixels. The standard gray pattern is not used by this routine; rather, the frame is drawn so that the top-left pixel of the frame is always on.

**See Also**
WinDrawRectangleFrame, WinEraseRectangleFrame, WinGetFramesRectangle, WinInvertRectangleFrame, WinPaintRectangleFrame

WinDrawInvertedChars

**Purpose**
Draw the specified characters inverted (background color) in the draw window.

**Declared In**
Window.h

**Prototype**
```c
void WinDrawInvertedChars (const Char *chars, Int16 len, Coord x, Coord y)
```

**Parameters**
- `chars` Pointer to the characters to draw.
- `len` Length in bytes of the characters to draw.
- `x` x coordinate of the first character to draw (left bound).
- -> y y coordinate of the first character to draw (top bound).

**Result**
Returns nothing.

**Comments**
This routine draws the on bits and any underline in the background color and the off bits in the text color. (Black and white uses copy NOT mode.) This is the standard function for drawing inverted text.

Before calling this function, consider calling *WinSetUnderlineMode* and *FntSetFont*.

**See Also**
*WinDrawChar*, *WinDrawChars*, *WinDrawTruncChars*, *WinEraseChars*, *WinInvertChars*, *WinPaintChar*, *WinPaintChars*

---

**WinDrawLine**

**Purpose**
Draw a line in the draw window using the current foreground color.

**Declared In**
Window.h

**Prototype**
```c
void WinDrawLine (Coord x1, Coord y1, Coord x2, Coord y2)
```

**Parameters**
- -> x1 x coordinate of line start point.
- -> y1 y coordinate of line start point.
- -> x2 x coordinate of line endpoint.
- -> y2 y coordinate of line endpoint.

**Result**
Returns nothing.

**Comments**
This function differs from *WinPaintLine* in that it always uses winPaint mode (see *WinDrawOperation*). WinPaintLine uses the current drawing state transfer mode instead of winPaint.

**See Also**
WinDrawPixel

**Purpose**
Draw a pixel in the draw window using the current foreground color.

**Declared In**
Window.h

**Prototype**
```c
void WinDrawPixel (Coord x, Coord y)
```

**Parameters**
- `x` Pointer to the x coordinate of a pixel.
- `y` Pointer to the y coordinate of a pixel.

**Result**
Returns nothing. May display a fatal error message if the draw window’s bitmap is compressed.

**Compatibility**
Implemented only if [3.5 New Feature Set](#) is present.

**See Also**
WinErasePixel, WinInvertPixel, WinPaintPixel, WinPaintPixels

WinDrawRectangle

**Purpose**
Draw a rectangle in the draw window using the current foreground color.

**Declared In**
Window.h

**Prototype**
```c
void WinDrawRectangle (const RectangleType *rP, UInt16 cornerDiam)
```

**Parameters**
- `rP` Pointer to the rectangle to draw.
- `cornerDiam` Radius of rounded corners. Specify zero for square corners.

**Result**
Returns nothing.
Comments
The cornerDiam parameter specifies the radius of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides.

This function differs from WinPaintRectangle in that it always uses winPaint mode (see WinDrawOperation). WinPaintRectangle uses the current drawing state transfer mode instead of winPaint.

See Also
WinEraseRectangle, WinFillRectangle, WinInvertRectangle

WinDrawRectangleFrame

Purpose
Draw a rectangular frame in the draw window using the current foreground color.

Declared In
Window.h

Prototype
void WinDrawRectangleFrame (FrameType frame, const RectangleType *rP)

Parameters
-> frame Type of frame to draw (see FrameType).
-> rP Pointer to the rectangle to frame.

Result
Returns nothing.

Comments
The frame is drawn outside the specified rectangle.

This function differs from WinPaintRectangleFrame in that it always uses winPaint mode (see WinDrawOperation). WinPaintRectangleFrame uses the current drawing state transfer mode instead of winPaint.

See Also
WinDrawGrayRectangleFrame, WinEraseRectangleFrame, WinGetFramesRectangle, WinInvertRectangleFrame
### WinDrawTruncChars

**Purpose**
Draw the specified characters in the draw window, truncating the characters to the specified width.

**Declared In**
Window.h

**Prototype**
```c
void WinDrawTruncChars (const Char *chars, Int16 len, Coord x, Coord y, Coord maxWidth)
```

**Parameters**
- `-> chars` Pointer to the characters to draw.
- `-> len` Length in bytes of the characters to draw.
- `-> x` x coordinate of the first character to draw (left bound).
- `-> y` y coordinate of the first character to draw (top bound).
- `-> maxWidth` Maximum width in pixels of the characters that are to be drawn.

**Result**
Returns nothing.

**Comments**
Before calling this function, consider calling `WinSetUnderlineMode` and `FntSetFont`.

If drawing all of the specified characters requires more space than `maxWidth` allows, `WinDrawTruncChars` draws one less than the number of characters that can fit in `maxWidth` and then draws an ellipsis (...) in the remaining space. (If the boundary characters are narrower than the ellipsis, more than one character may be dropped to make room.) If `maxWidth` is narrower than the width of an ellipsis, nothing is drawn.

Use this function to truncate text that may contain multi-byte characters.

**Compatibility**
Implemented only if 3.1 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call
WinGlueDrawTruncChars. For more information, see Chapter 75, “PalmOSGlue Library.”

See Also: WinDrawChar, WinDrawChars, WinDrawInvertedChars, WinEraseChars, WinInvertChars, WinPaintChar, WinPaintChars

WinEraseChars

Purpose: Erase the specified characters in the draw window.

Declared In: Window.h

Prototype: void WinEraseChars (const Char *chars, Int16 len, Coord x, Coord y)

Parameters:
- chars: Pointer to the characters to erase.
- len: Length in bytes of the characters to erase.
- x: x coordinate of the first character to erase (left bound).
- y: y coordinate of the first character to erase (top bound).

Result: Returns nothing.

Comments: The winMask transfer mode is used to erase the characters. See WinDrawOperation for more information. This has the effect of erasing only the on bits for the characters rather than the entire text rectangle. This function only works if the foreground color is black and the background color is white.

See Also: WinDrawChar, WinDrawChars, WinDrawInvertedChars, WinDrawTruncChars, WinInvertChars, WinPaintChar, WinPaintChars
WinEraseLine

**Purpose**  
Draw a line in the draw window using the current background color.

**Declared In**  
Window.h

**Prototype**  
void WinEraseLine (Coord x1, Coord y1, Coord x2, Coord y2)

**Parameters**
- x1  
x coordinate of line start point.
- y1  
y coordinate of line start point.
- x2  
x coordinate of line endpoint.
- y2  
y coordinate of line endpoint.

**Result**  
Returns nothing.

**See Also**  

WinErasePixel

**Purpose**  
Draw a pixel in the draw window using the current background color.

**Declared In**  
Window.h

**Prototype**  
void WinErasePixel (Coord x, Coord y)

**Parameters**
- x  
Pointer to the x coordinate of a pixel.
- y  
Pointer to the y coordinate of a pixel.

**Result**  
Returns nothing.
WinEraseRectangle

Purpose
Draw a rectangle in the draw window using the current background color.

Declared In
Window.h

Prototype
void WinEraseRectangle (const RectangleType *rP, UInt16 cornerDiam)

Parameters
- rP  Pointer to the rectangle to erase.
- cornerDiam  Radius of rounded corners. Specify zero for square corners.

Result
Returns nothing.

Comments
The cornerDiam parameter specifies the radius of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides.

See Also
WinDrawRectangle, WinFillRectangle, WinInvertRectangle, WinPaintRectangle
WinEraseRectangleFrame

**Purpose**
Draw a rectangular frame in the draw window using the current background color.

**Declared In**
Window.h

**Prototype**
```c
void WinEraseRectangleFrame (FrameType frame, const RectangleType *rP)
```

**Parameters**
- `-> frame` Type of frame to draw (see `FrameType`).
- `-> rP` Pointer to the rectangle to frame.

**Result**
Returns nothing.

**See Also**
WinDrawGrayRectangleFrame, WinDrawRectangleFrame, WinGetFramesRectangle, WinInvertRectangleFrame, WinPaintRectangleFrame

WinEraseWindow

**Purpose**
Erase the contents of the draw window.

**Declared In**
Window.h

**Prototype**
```c
void WinEraseWindow (void)
```

**Parameters**
None.

**Result**
Returns nothing.

**Comments**
WinEraseRectangle is used to erase the window. This routine doesn’t erase the frame around the draw window. See WinEraseRectangleFrame and WinGetWindowFrameRect.
**WinFillLine**

**Purpose**  Fill a line in the draw window with the current pattern.

**Declared In**  Window.h

**Prototype**  
```c
void WinFillLine (Coord x1, Coord y1, Coord x2, Coord y2)
```

**Parameters**
- `x1`  x coordinate of line start point.
- `y1`  y coordinate of line start point.
- `x2`  x coordinate of line endpoint.
- `y2`  y coordinate of line endpoint.

**Result**  Returns nothing.

**Comments**  You can set the current pattern with `WinSetPattern`.

**See Also**  `WinDrawGrayLine`, `WinDrawLine`, `WinEraseLine`, `WinInvertLine`, `WinPaintLine`, `WinPaintLines`

---

**WinFillRectangle**

**Purpose**  Draw a rectangle in the draw window with current pattern.

**Declared In**  Window.h

**Prototype**  
```c
void WinFillRectangle (const RectangleType *rP, UInt16 cornerDiam)
```

**Parameters**
- `rP`  Pointer to the rectangle to draw.
- `cornerDiam`  Radius of rounded corners. Specify zero for square corners.

**Result**  Returns nothing.
Comments  You can set the current pattern with WinSetPattern.
The cornerDiam parameter specifies the radius of four imaginary
circles used to form the rounded corners. An imaginary circle is
placed within each corner tangent to the rectangle on two sides.

See Also  WinDrawRectangle, WinEraseRectangle,
WinInvertRectangle, WinPaintRectangle

WinGetActiveWindow

Purpose  Return the window handle of the active window.

Declared In  Window.h

Prototype  WinHandle WinGetActiveWindow (void)

Parameters  None.

Result  Returns the handle of the active window. All user input is directed
to the active window.

See Also  WinSetActiveWindow, WinGetDisplayWindow,
WinGetFirstWindow, WinGetDrawWindow

WinGetBitmap

Purpose  Return a pointer to a window’s bitmap, which holds the window
contents.

Declared In  Window.h

Prototype  BitmapType *WinGetBitmap (WinHandle winHandle)

Parameters  ~> winHandle  Handle of window from which to get the
bitmap.

Result  Returns a pointer to the bitmap or NULL if winHandle is invalid.
Comments
For onscreen windows, the bitmap returned always represents the whole screen. Thus, the top-left corner of the returned bitmap may not be the top-left corner of the window.

Compatibility
Implemented only if 3.5 New Feature Set is present.

WinGetBounds

Purpose
Return the bounds of the current draw window in display-relative coordinates.

Declared In
Window.h

Prototype
void WinGetBounds (WinHandle winH, RectangleType *rP)

Parameters
-> winH Handle to a window.
<- rP Pointer to a rectangle.

Result
Returns nothing.

Comments
This function returns in rP the bounds of the window represented by winH. This corresponds to the convention used by WinSetBounds, because it takes a window handle as an argument.

Prior to Palm OS 4.0, WinGetBounds returned the bounds of the draw window, and did not take a window handle as an argument. If an application needed to determine the bounds of an arbitrary window, the application would call WinSetDrawWindow to temporarily set the draw window to the desired window, then WinGetBounds would be called to get the bounds of the draw window, then WinSetDrawWindow would be called again to restore the draw window. This is no longer necessary.

Compatibility
Implemented only if 4.0 New Feature Set is not present. As of Palm OS 4.0, applications should use WinGetDrawWindowBounds.

See Also
WinGetWindowExtent, WinGetDrawWindowBounds
WinGetClip

**Purpose**  Return the clipping rectangle of the draw window.

**Declared In**  Window.h

**Prototype**  void WinGetClip (RectangleType *rP)

**Parameters**  <- rP  Pointer to a structure to hold the clipping bounds.

**Result**  Returns nothing.

**See Also**  WinSetClip

\[\frown\]

New  WinGetCoordinateSystem

**Purpose**  Get the coordinate system

**Declared In**  Window.h

**Prototype**  UInt16 WinGetCoordinateSystem (void)

**Parameters**  None.

**Result**  Returns a value representing the current coordinate system. See “Window Coordinate System Constants” on page 1163 for the values that this function can return.

**Comments**  Use this function to determine the active coordinate system. Armed with this information, an application can properly initialize graphic primitive coordinates and dimensions, or can modify the coordinate system with WinSetCoordinateSystem.

**Compatibility**  Implemented only if the High-Density Display Feature Set is
present.

**See Also**  
[WinSetCoordinateSystem](#)

### WinGetDisplayExtent

**Purpose**  
Return the width and height of the display (the screen).

**Declared In**  
Window.h

**Prototype**  
```c
void WinGetDisplayExtent (Coord *extentX,
Coord *extentY)
```

**Parameters**  
<- extentX  
Pointer to the width of the display in pixels.

<- extentY  
Pointer to the height of the display in pixels.

**Result**  
Returns nothing.

### WinGetDisplayWindow

**Purpose**  
Return the window handle of the display (screen) window.

**Declared In**  
Window.h

**Prototype**  
```c
WinHandle WinGetDisplayWindow (void)
```

**Parameters**  
None.

**Result**  
Returns the handle of display window.

**Comments**  
The display window is created by the system at start-up; it has the same size as the Palm OS drawable area of the physical display (screen).

**See Also**  
[WinGetDisplayExtent](#), [WinGetActiveWindow](#), [WinGetDrawWindow](#)
WinGetDrawWindow

Purpose Return the window handle of the current draw window.

Declared In Window.h

Prototype WinHandle WinGetDrawWindow (void)

Parameters None.

Result Returns handle of draw window.

See Also WinGetDisplayWindow, WinGetActiveWindow, WinSetDrawWindow

WinGetDrawWindowBounds

Purpose Return the bounds of the draw window.

Declared In Window.h

Prototype void WinGetDrawWindowBounds (RectangleType *rP)

Parameters <- rP Pointer to the window bounds.

Result Returns nothing.

Comments A pointer to the bounds of the draw window is returned. This function is equivalent to WinGetBounds that was in Palm OS prior to 4.0.

See Also WinGetBounds
**WinGetFirstWindow**

**Purpose**
Return a pointer to the first window in the linked list of windows.

**Declared In**
Window.h

**Prototype**
```c
WinHandle WinGetFirstWindow (void)
```

**Parameters**
None.

**Result**
Returns handle of first window.

**Comments**
This function is usually used by the system only.

**See Also**
WinGetActiveWindow

**WinGetFramesRectangle**

**Purpose**
Return the rectangle that includes a rectangle together with the specified frame around it.

**Declared In**
Window.h

**Prototype**
```c
void WinGetFramesRectangle (FrameType frame, const RectangleType *rP, RectangleType *obscuredRect)
```

**Parameters**
- `-> frame` Type of rectangle frame (see `FrameType`).
- `-> rP` Pointer to the rectangle to frame.
- `<- obscuredRect` Pointer to the rectangle that includes both the specified rectangle and its frame.

**Result**
Returns nothing.
Comments
Frames are always drawn around (outside) a rectangle.

See Also
WinGetWindowFrameRect, WinGetBounds

WinGetPattern

Purpose
Return the current fill pattern.

Declared In
Window.h

Prototype
void WinGetPattern (CustomPatternType *patternP)

Parameters
<- patternP Buffer where the current pattern is returned (see CustomPatternType).

Result
Returns nothing.

Comments
The fill pattern is used by WinFillLine and WinFillRectangle. This function returns the value of patternData in the current drawing state. (See DrawStateType.) The patternData field is only set if the pattern field is customPattern. Therefore, it’s a good idea to use WinGetPatternType instead of this function on systems that support WinGetPatternType.

See Also
WinSetPattern
WinGetPatternType

Purpose
Return the current pattern type.

Declared In
Window.h

Prototype
PatternType WinGetPatternType (void)

Parameters
None.

Result
Returns the current draw window pattern type (see PatternType). If the return value is customPattern, you can retrieve the pattern with WinGetPattern.

Comments
The fill pattern is used by WinFillLine and WinFillRectangle.

Compatibility
Implemented only if 3.5 New Feature Set is present.

See Also
WinSetPatternType

WinGetPixel

Purpose
Return the color value of a pixel in the current draw window.

Declared In
Window.h

Prototype
IndexedColorType WinGetPixel (Coord x, Coord y)

Parameters
- x Pointer to the x coordinate of a pixel.
- y Pointer to the y coordinate of a pixel.

Result
Returns the indexed color value of the pixel. See IndexedColorType. A return value of 0 means either that the coordinates do not lie in the current draw window or that they do and the color of that pixel is index 0 (typically white).
WinGetPixelRGB

Purpose
Return the RGB color values of a pixel in the current draw window.

Declared In
Window.h

Prototype
Err WinGetPixelRGB (Coord x, Coord y, RGBColorType *rgbP)

Parameters
- x
  Pointer to the x coordinate of a pixel.
- y
  Pointer to the y coordinate of a pixel.
- rgbP
  RGB color components of the pixel.

Result
Returns errNone or sysErrParamErr. sysErrParamErr is returned when the x or y arguments are < 0 or when they are outside the bounds of the draw window.

Comments
The RGB color values of the pixel are returned as an RGBColorType. This function can be used with both indexed or direct color modes. A return value of sysErrParamErr means that the coordinates do not lie in the current draw window.

Compatibility
Implemented only if 4.0 New Feature Set is present.
**New**

**WinGetSupportedDensity**

**Purpose**
Enumerate the various display densities supported by the blitter.

**Declared In**
Window.h

**Prototype**
Err WinGetSupportedDensity (UInt16 *densityP)

**Parameters**
- `<-> densityP` Pointer to a supported density value. Set this value to zero before calling this function for the first time. Subsequent calls cause this value to be set to one of the display densities supported by the handheld.

**Result**
Returns errNone unless the value you supply in *densityP isn’t a supported density and isn’t zero, in which case this function returns SysErrParamErr.

**Comments**
Initialize *densityP to zero before your application calls this function for the first time. Repeated calls to WinGetSupportedDensity will cause the value pointed to by densityP to change; these values represent the supported display densities, in order from low to high density. After the last supported density value, this function sets *densityP back to zero.

**NOTE:** The densities reported by this function are those that are supported by the blitter. These densities are not necessarily supported by the underlying hardware. A handheld with a low-density screen that is able to scale high-density bitmaps will report that it can handle both high and low density bitmaps. Use WinScreenGetAttribute to determine the density of the handheld’s screen.

Density values are defined in Bitmap.h; see the DensityType enum. Only those values supported by a given handheld will be returned by WinGetSupportedDensity. For example, on a...
handheld with a double-density display this function returns
kDensityLow, followed by kDensityDouble, followed by 0. For
each supported density, the inverse scaling factor is supported. In
this example, the blitter supports pixel-doubling low-density data
for a double-density destination, and the blitter supports pixel-
halving high-density data for a low-density destination.

The value pointed to by densityP should only be zero or one of
the density values supported by the handheld. If it has any other
value when you call WinGetSupportedDensity, this function
will simply return sysErrParamErr.

**Compatibility**

Implemented only if the [High-Density Display Feature Set](#) is
present.

**WinGetWindowExtent**

**Purpose**
Return the width and height of the current draw window.

**Declared In**
Window.h

**Prototype**
```c
void WinGetWindowExtent (Coord *extentX,
Coord *extentY)
```

**Parameters**
- `- extentX` Pointer to the width in pixels of the draw
  window.
- `- extentY` Pointer to the height in pixels of the draw
  window.

**Result**
Returns nothing.

**See Also**
[WinGetBounds](#), [WinGetWindowFrameRect](#)
**WinGetWindowFrameRect**

**Purpose**
Return a rectangle, in display-relative coordinates, that defines the size and location of a window and its frame.

**Declared In**
Window.h

**Prototype**
```c
void WinGetWindowFrameRect (WinHandle winHandle, RectangleType *r)
```

**Parameters**
- `winHandle`
  Handle of window whose coordinates are desired.
- `r`
  Pointer to the coordinates of the window.

**Result**
Returns nothing.

**See Also**
WinGetBounds

**WinIndexToRGB**

**Purpose**
Convert an index in the currently active color table to an RGB value.

**Declared In**
Window.h

**Prototype**
```c
void WinIndexToRGB (IndexedColorType i, RGBColorType *rgbP)
```

**Parameters**
- `i`
  A color index value. See IndexedColorType.
- `rgbP`
  Pointer to an RGB color value corresponding to the index value i. See RGBColorType.

**Result**
Returns nothing.

**Compatibility**
Implemented only if 3.5 New Feature Set is present.

**See Also**
WinRGBToIndex
**WinInvertChars**

**Purpose**
Invert the specified characters in the draw window.

**Declared In**
Window.h

**Prototype**
```c
void WinInvertChars (const Char *chars, Int16 len, Coord x, Coord y)
```

**Parameters**
- `-> chars` Pointer to the characters to invert.
- `-> len` Length in bytes of the characters to invert.
- `-> x` x coordinate of the first character to invert (left bound).
- `-> y` y coordinate of the first character to invert (top bound).

**Result**
Returns nothing.

**Comments**
This function applies the winInvert operation of WinDrawOperation to the characters in the draw window.

To perform color inverting, use WinSetTextColor and WinSetBackColor to choose the desired colors, and call WinPaintChar.

**See Also**
WinDrawChar, WinDrawChars, WinDrawInvertedChars, WinDrawTruncChars, WinEraseChars, WinPaintChar, WinPaintChars
WinInvertLine

**Purpose**
Invert a line in the draw window (using the WinDrawOperation winInvert).

**Declared In**
Window.h

**Prototype**
void WinInvertLine (Coord x1, Coord y1, Coord x2, Coord y2)

**Parameters**
- x1  
  x coordinate of line start point.
- y1  
  y coordinate of line start point.
- x2  
  x coordinate of line endpoint.
- y2  
  y coordinate of line endpoint.

**Result**
Returns nothing.

**See Also**

WinInvertPixel

**Purpose**
Invert a pixel in the draw window (using the WinDrawOperation winInvert).

**Declared In**
Window.h

**Prototype**
void WinInvertPixel (Coord x, Coord y)

**Parameters**
- x  
  Pointer to the x coordinate of a pixel.
- y  
  Pointer to the y coordinate of a pixel.

**Result**
Returns nothing.
Compatibility  Implemented only if 3.5 New Feature Set is present.

See Also  WinDrawPixel, WinErasePixel, WinPaintPixel, WinPaintPixels

WinInvertRectangle

Purpose  Invert a rectangle in the draw window (using the WinDrawOperation winInvert).

Declared In  Window.h

Prototype  void WinInvertRectangle (const RectangleType *rP, UInt16 cornerDiam)

Parameters  -> rP  Pointer to the rectangle to invert.
            -> cornerDiam  Radius of rounded corners. Specify zero for square corners.

Result  Returns nothing.

Comments  The cornerDiam parameter specifies the radius of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides.

The operating system itself does not use the inverting routines. Instead, it uses the winSwap transfer mode, or it changes the color selection and uses the WinPaint... routines.

See Also  WinDrawRectangle, WinEraseRectangle, WinFillRectangle, WinPaintRectangle
**WinInvertRectangleFrame**

**Purpose**
Invert a rectangular frame in the draw window (using the WinDrawOperation winInvert).

**Declared In**
Window.h

**Prototype**
void WinInvertRectangleFrame (FrameType frame, const RectangleType *rP)

**Parameters**
- `frame` Type of frame to draw (see FrameType).
- `rP` Pointer to the rectangle to frame.

**Result**
Returns nothing.

**See Also**
WinDrawGrayRectangleFrame, WinDrawRectangleFrame, WinEraseRectangleFrame, WinGetFramesRectangle, WinPaintRectangleFrame

**WinModal**

**Purpose**
Return true if the specified window is modal.

**Declared In**
Window.h

**Prototype**
Boolean WinModal (WinHandle winHandle)

**Parameters**
- `winHandle` Handle of a window.

**Result**
Returns true if the window is modal, otherwise false.

**Comments**
A window is modal if it cannot lose the focus.

**See Also**
FrmAlert, FrmCustomAlert, FrmDoDialog
WinPaintBitmap

**Purpose**
Draw a bitmap in the current draw window at the specified coordinates with the current draw mode.

**Declared In**
Window.h

**Prototype**
```c
void WinPaintBitmap (BitmapType *bitmapP,
                     Coord x, Coord y)
```

**Parameters**
- `-> bitmapP` Pointer to a bitmap.
- `-> x` The x coordinate of the top-left corner.
- `-> y` The y coordinate of the top-left corner.

**Result**
Returns nothing.

**Comments**
If the bitmap has multiple depths (is a bitmap family), the closest match less than or equal to the current draw window depth is used. If such a bitmap does not exist, the bitmap with the closest match greater than the draw window depth is used.

Using `WinPaintBitmap` is now recommended instead of the previous practice of rendering bitmaps into an offscreen window and then using `WinCopyRectangle` to draw them on screen.

The current draw mode is set by `WinSetDrawMode`.

If the bitmap has its own color table, color conversion to the draw window color table will be applied (on OS 3.5 or later). This color conversion is slow and not recommended. Instead of including a color table in the bitmap, consider using `WinPalette` to change the system color table, draw the bitmap, and then change the system color table back when the bitmap is no longer visible.

**Compatibility**
Implemented only if 3.5 New Feature Set is present.

**See Also**
`WinDrawBitmap`, `WinEraseRectangle`, `WinPaintTiledBitmap`
WinPaintChar

Purpose
Draw a character in the draw window using the current drawing state.

Declared In
Window.h

Prototype
void WinPaintChar (WChar theChar, Coord x, Coord y)

Parameters
-> theChar
The character to draw. This may be either a single-byte character or a multi-byte character.

-> x
x coordinate of the location where the character is to be drawn (left bound).

-> y
y coordinate of the location where the character is to be drawn (top bound).

Result
Returns nothing.

See Also
WinPaintChar draws the on bits in the text color and the off bits in the background color, with underlines (if any) drawn in the foreground color using the current drawing mode.

This function uses the current drawing state, which is stored in a DrawStateType structure. See the description of that structure to learn the functions you can call to set the drawing state to the values you want.

Compatibility
Implemented only if 3.5 New Feature Set is present.

See Also
WinDrawChar, WinDrawChars, WinDrawInvertedChars, WinDrawTruncChars, WinEraseChars, WinInvertChars, WinPaintChars
WinPaintChars

Purpose  Draw the specified characters in the draw window with the current draw state.

Declared In  Window.h

Prototype  void WinPaintChars (const Char *chars, Int16 len, Coord x, Coord y)

Parameters

- -> chars  Pointer to the characters to draw.
- -> len  Length in bytes of the characters to draw.
- -> x  x coordinate of the first character to draw (left bound).
- -> y  y coordinate of the first character to draw (top bound).

Result  Returns nothing.

Comments  WinPaintChars draws the on bits in the text color and the off bits in the background color, with underlines (if any) drawn in the foreground color using the current drawing mode.

This function uses the current drawing state, which is stored in a DrawStateType structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

Before calling this function, consider calling WinSetUnderlineMode and FntSetFont.

Compatibility  Implemented only if 3.5 New Feature Set is present.

See Also  WinDrawChar, WinDrawChars, WinDrawInvertedChars, WinDrawTruncChars, WinEraseChars, WinInvertChars, WinPaintChar
WinPaintLine

**Purpose**
Draw a line in the draw window using the current drawing state.

**Declared In**
Window.h

**Prototype**
void WinPaintLine (Coord x1, Coord y1, Coord x2, Coord y2)

**Parameters**
- x1 x coordinate of line beginning point.
- y1 y coordinate of line beginning point.
- x2 x coordinate of line endpoint.
- y2 y coordinate of line endpoint.

**Result**
Returns nothing.

**Comments**
This function uses the current drawing state, which is stored in a DrawStateType structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

**Compatibility**
Implemented only if 3.5 New Feature Set is present.

**See Also**

WinPaintLines

**Purpose**
Draw several lines in the draw window using the current drawing state.

**Declared In**
Window.h

**Prototype**
void WinPaintLines (UInt16 numLines, WinLineType lines[])

**Parameters**
- numLines Number of lines to paint.
WinPaintPixel

Purpose
Render a pixel in the draw window using the current drawing state.

Declared In
Window.h

Prototype
void WinPaintPixel (Coord x, Coord y)

Parameters
- x
  Pointer to the x coordinate of a pixel.
- y
  Pointer to the y coordinate of a pixel.

Result
Returns nothing.

Comments
This function uses the current drawing state, which is stored in a
DrawStateType structure. See the description of that structure to
learn the functions you can call to set the drawing state to the state
you want.

Compatibility
Implemented only if 3.5 New Feature Set is present.

See Also
WinDrawPixel, WinErasePixel, WinInvertPixel, WinPaintPixel, WinPaintPixels
WinPaintPixels

**Purpose**
Render several pixels in the draw window using the current drawing state.

**Declared In**
Window.h

**Prototype**
```c
void WinPaintPixels (UInt16 numPoints, PointType pts[])
```

**Parameters**
- `numPoints` Number of pixels to paint.
- `pts` Array of pixels.

**Result**
Returns nothing.

**Comments**
This function uses the current drawing state, which is stored in a `DrawStateType` structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

**Compatibility**
Implemented only if 3.5 New Feature Set is present.

**See Also**
WinDrawPixel, WinErasePixel, WinInvertPixel, WinPaintPixel

WinPaintRectangle

**Purpose**
Draw a rectangle in the draw window using the current drawing state.

**Declared In**
Window.h

**Prototype**
```c
void WinPaintRectangle (const RectangleType *rP, UInt16 cornerDiam)
```

**Parameters**
- `rP` Pointer to the rectangle to draw.
cornerDiam

Radius of rounded corners. Specify zero for square corners.

Result

Returns nothing.

Comments

The cornerDiam parameter specifies the radius of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides. This function uses the current drawing state, which is stored in a DrawStateType structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

Compatibility

Implemented only if 3.5 New Feature Set is present.

See Also

WinDrawRectangle, WinEraseRectangle, WinFillRectangle, WinInvertRectangle

WinPaintRectangleFrame

Purpose

Draw a rectangular frame in the draw window using the current drawing state.

Declared In

Window.h

Prototype

void WinPaintRectangleFrame (FrameType frame, const RectangleType *rP)

Parameters

-> frame Type of frame to draw (see FrameType).
-> rP Pointer to the rectangle to frame.

Result

Returns nothing.

Comments

The frame is drawn outside the specified rectangle. This function uses the current drawing state, which is stored in a DrawStateType structure. See the description of that structure to
learn the functions you can call to set the drawing state to the state you want.

**Compatibility**

Implemented only if 3.5 New Feature Set is present.

**See Also**

WinDrawGrayRectangleFrame, WinDrawRectangleFrame, WinEraseRectangleFrame, WinGetFramesRectangle, WinInvertRectangleFrame, WinPaintRoundedRectangleFrame

---

**New**

**WinPaintRoundedRectangleFrame**

**Purpose**

Draw a rectangular frame with rounded corners in the draw window using the current drawing state.

**Declared In**

Window.h

**Prototype**

```c
void WinPaintRoundedRectangleFrame
(const RectangleType *rP, Coord width,
Coord cornerRadius, Coord shadowWidth)
```

**Parameters**

- `rP` Pointer to the rectangle to frame.
- `width` The width of the frame, interpreted using the active coordinate system.
- `cornerRadius` The radius of the rectangle’s rounded corners, interpreted using the active coordinate system.
- `shadowWidth` The shadow offset, interpreted using the active coordinate system.

**Result**

Returns nothing.

**Comments**

This function allows you to draw a rectangle with a frame width and corner radius specified in the active coordinate system. It is necessary because WinPaintRectangleFrame doesn’t allow you to draw rounded rectangles with a frame width greater than 2. Note that because there isn’t a function that parallels either
WinDrawRectangleFrame, WinEraseRectangleFrame, or WinInvertRectangleFrame, you must set the drawing mode and colors as appropriate and use WinPaintRoundedRectangleFrame to achieve the desired effect.

**Compatibility**

Implemented only if the High-Density Display Feature Set is present.

---

**New**

**WinPaintTiledBitmap**

**Purpose**

Fill a rectangle with a pattern defined by a bitmap.

**Declared In**

Window.h

**Prototype**

```c
void WinPaintTiledBitmap (BitmapType *bitmapP, RectangleType *rectP)
```

**Parameters**

- `-> bitmapP` Pointer to the bitmap that contains the desired pattern.
- `-> rectP` Pointer to the rectangle that is to be filled.

**Result**

Returns nothing. On a debug ROM, if either `bitmapP` or `rectP` are NULL, an error is displayed.

**Comments**

This function makes it possible for an application to define a pattern that is larger than the standard 8 by 8 custom pattern, and to define high-density custom patterns.

The pattern is scaled by the blitter using the density of `bitmapP` and the density of the screen bitmap. `bitmapP` can be a bitmap family; if it is, the Window Manager selects a bitmap using the same algorithm used by `WinPaintBitmap`. As with other patterns, the tiled pattern is anchored to the window’s origin.

If `bitmapP` does not match the depth or density of the destination bitmap, the blitter converts the bitmap using a temporary buffer.
Note that if there isn’t enough heap space for the temporary buffer, WinPaintTiledBitmap will be slow.

Compatibility  Implemented only if the High-Density Display Feature Set is present.

WinPalette

Purpose  Set or retrieve the palette for the draw window.

Declared In  Window.h

Prototype  Err WinPalette (UInt8 operation,
Int16 startIndex, UInt16 paletteEntries,
RGBColorType *tableP)

Parameters  -> operation  Specify one of the following values:
  winPaletteGet
  Retrieve the palette. Entries are read from the palette beginning at startIndex and placed into tableP beginning at index 0.
  winPaletteSet
  Set the palette. Entries from tableP (beginning at index 0) are set into the palette beginning at startIndex in the palette.
  winPaletteSetToDefault
  Set the palette to the default system palette.
> startIndex Identifies where in the palette to start reading or writing. Specify WinUseTableIndexes to indicate that the entries are not to be set or read sequentially; instead, the index value in each RGBColorType entry in tableP determines which slot in the palette is to be set or read. You can use this technique to get or set several discontiguous palette entries with a single function call.

> paletteEntries Number of palette entries to get or set.

<-> tableP A pointer to a buffer of RGBColorType entries that is either read from or written to, depending on the operation parameter; the table entries from 0 to paletteEntries – 1 are affected by this routine.

**Result** Returns one of the following values:

- errNone Success.
- winErrPalette The current draw window does not have a color table, a set operation has overflowed the color table, or one of the entries in tableP has an invalid index value.
- sysErrMsgParamErr The startIndex value is invalid.

**Comments** Here are some examples of how this routine works:

- If startIndex is 0 and paletteEntries is 10, the first 10 elements of the palette will be set from tableP or will be copied into tableP.
- If startIndex is 10 and paletteEntries is 5, then entries 10, 11, 12, 13, and 14 in the palette will be set from or copied to elements 0, 1, 2, 3, and 4 in tableP.
- If startIndex is WinUseTableIndexes and paletteEntries is 1, then the index value in the RGBColorType of element 0 of tableP will be read from or copied to tableP; in this case, the index field of the RGBColorType will not change.
During a set operation, this function broadcasts the `sysNotifyDisplayChangeEvent` to notify any interested observer that the color palette has changed.

One use for this function is if you need to display a bitmap that uses a color table other than the one in use by the system. You can attach a custom color table to a bitmap, and if you do, the bitmap is drawn using that color table. However, this is a performance drain. As an optimization, you can use `WinPalette` to change the system color table to match that used by the bitmap, display the bitmap, and use `WinPalette` to reset the color table when the bitmap is no longer visible.

**Compatibility**
Implemented only if [3.5 New Feature Set](#) is present.

---

## WinPopDrawState

**Purpose**
Restore the draw state values to the last saved set on the stack.

**Declared In**
Window.h

**Prototype**
```c
void WinPopDrawState (void)
```

**Parameters**
None.

**Result**
Returns nothing.

**Comments**
Use this routine to restore the draw state saved by the previous call to `WinPushDrawState`.

After you call this function, the current draw window’s `drawStateP` field points to the restored drawing state.

**Compatibility**
Implemented only if [3.5 New Feature Set](#) is present.
**WinPushDrawState**

**Purpose**
Save the current draw state values onto the draw state stack.

**Declared In**
Window.h

**Prototype**
void WinPushDrawState (void)

**Parameters**
None.

**Result**
Returns nothing.

**Comments**
Use this routine to save the current draw state before making changes to it using the functions listed in the DrawStateType structure’s description. Call WinPopDrawState to restore the saved settings.

**Compatibility**
Implemented only if 3.5 New Feature Set is present.

**WinResetClip**

**Purpose**
Reset the clipping rectangle of the draw window to the portion of the draw window that is within the bounds of the display.

**Declared In**
Window.h

**Prototype**
void WinResetClip (void)

**Parameters**
None.

**Result**
Returns nothing.

**See Also**
WinSetClip
WinRestoreBits

**Purpose**
Copy the contents of the specified window to the draw window and delete the passed window.

**Declared In**
Window.h

**Prototype**
void WinRestoreBits (WinHandle winHandle, Coord destX, Coord destY)

**Parameters**
- winHandle
  Handle of window to copy and delete.
- destX
  x coordinate in the draw window to copy to.
- destY
  y coordinate in the draw window to copy to.

**Result**
Returns nothing.

**Comments**
This routine is generally used to restore a region of the display that was saved with WinSaveBits.

**See Also**
WinSaveBits

WinRGBToIndex

**Purpose**
Convert an RGB value to the index of the closest color in the currently active color lookup table (CLUT).

**Declared In**
Window.h

**Prototype**
IndexedColorType WinRGBToIndex (const RGBColorType *rgbP)

**Parameters**
- rgbP
  Pointer to an RGB color value.

**Result**
Returns the index of the closest matching color in the CLUT.

**Comments**
Palm OS 3.5 supports a maximum of 256 colors. The number of possible RGB colors greatly exceeds this amount. For this reason, an
exact match may not be available for rgbP. If there is no exact RGB match, then a luminance best-fit is used if the color lookup table is entirely gray scale (red, green, and blue values for each entry are identical), or a shortest-distance fit in RGB space is used if the palette contains colors. RGB shortest distance may not always produce the actual closest perceptible color, but it’s relatively fast and works for the system palette.

WinRGBToIndex uses the draw window’s color table to return the appropriate color table index. If the draw window does not have a color table, the default color table of the current screen is used.

If the draw window does not have a color table, and if the depth of the draw window and the depth of the screen are different, this function will return an inappropriate index. If this situation exists, the application should either define a color table for the draw window, or use WinScreenMode to set the screen depth to the same depth as the draw window before calling WinRGBToIndex.

**NOTE:** The bitmap data will not be blitted properly if the depth of the screen is changed using WinScreenMode and the new window uses a bitmap that does not define the bitmap’s color table. See WinScreenMode for information on how to work around this limitation.

**Compatibility**

Implemented only if 3.5 New Feature Set is present.

**See Also**

WinIndexToRGB, WinScreenMode
WinSaveBits

Purpose Create an offscreen window and copy the specified region from the draw window to the offscreen window.

Declared In Window.h

Prototype WinHandle WinSaveBits (const RectangleType *source, UInt16 *error)

Parameters
- source Pointer to the bounds of the region to save, relative to the display.
- error Pointer to any error encountered by this function.

Result Returns the handle of the window containing the saved image, or zero if an error occurred.

Comments The offscreen window is the same size as the region to copy.
This function tries to copy the window’s bitmap using compressed format if possible. It may display a fatal error message if an error occurs when it tries to shrink the pointer for the compressed bits.

See Also WinRestoreBits
New **WinScaleCoord**

**Purpose** Convert a single coordinate from the standard coordinate system to the active coordinate system.

**Declared In** Window.h

**Prototype**

```c
Coord WinScaleCoord (Coord coord, Boolean ceiling)
```

**Parameters**

- `-> coord` A coordinate in the standard coordinate system.
- `-> ceiling` Pass `true` to round up, `false` to truncate the fractional part when scaling.

**Result** Returns the coordinate scaled to the active coordinate system.

**Comments**

This function converts a coordinate by multiplying it by the coordinate scaling factor, and then truncating or rounding the result to an integer value depending on the value of `ceiling`.

If the active coordinate system is `kCoordinatesStandard`, the returned coordinate is equal to the supplied coordinate.

**Compatibility** Implemented only if the [High-Density Display Feature Set](#) is present.

**See Also** [WinScalePoint](#), [WinScaleRectangle](#), [WinUnscaleCoord](#)
New

**WinScalePoint**

**Purpose**  Convert a point from the standard coordinate system to the active coordinate system.

**Declared In**  Window.h

**Prototype**  void WinScalePoint (PointType *pointP, Boolean ceiling)

**Parameters**

- `<> pointP`  Pointer to a PointType structure that, before the call, should contain a point’s standard coordinate system coordinates. After this function is called the PointType structure contains the coordinates of the point scaled to the active coordinate system.

- `-> ceiling`  Pass `true` to round up, `false` to truncate the fractional part when scaling.

**Result**  Returns nothing. The coordinates of the point indicated by `pointP` are converted to the active coordinate system.

**Comments**  This function converts a point by multiplying its x and y coordinates by the coordinate scaling factor and then truncating or rounding the results depending on the value of `ceiling`. If the active coordinate system is kCoordinatesStandard, `pointP` is not changed by this function.

**Compatibility**  Implemented only if the High-Density Display Feature Set is present.

**See Also**  WinScaleCoord, WinScaleRectangle, WinUnscalePoint
New **WinScaleRectangle**

**Purpose**
Convert a rectangle from the standard coordinate system to the active coordinate system.

**Declared In**
Window.h

**Prototype**
void WinScaleRectangle (RectangleType *rectP)

**Parameters**
<- rectP
Pointer to a RectangleType structure that, before the call, should contain a rectangle’s standard coordinate system coordinates. After this function is called the RectangleType structure contains the coordinates of the rectangle scaled to the active coordinate system.

**Result**
Returns nothing. The coordinates of the rectangle indicated by rectP are converted to the native coordinate system.

**Comments**
This function scales the rectangle’s topLeft and extent points by multiplying their x and y coordinates by the coordinate scaling factor. All values are then truncated, but if either topLeft.x or extent.x had a fractional part, extent.x is incremented by 1 (and, similarly, if either topLeft.y or extent.y had a fractional part, extent.y is incremented by 1).

If the active coordinate system is kCoordinatesStandard, rectP is not changed by this function.

You can use this function when your gadget handler draws using a more precise coordinate system than the Form Manager and needs to convert the form-based bounds of the gadget to the high-density bounds used by the gadget’s drawing function.

**Compatibility**
Implemented only if the [High-Density Display Feature Set](#) is
present.

See Also  WinScaleCoord, WinScalePoint, WinUnscaleRectangle

▼

New  WinScreenGetAttribute

Purpose  Get various attributes of the screen.

Declared In  Window.h

Prototype  Err WinScreenGetAttribute
            (WinScreenAttrType selector, UInt32 *attrP)

Parameters  -> selector  A value indicating which attribute to return.
            See the description of WinScreenAttrType in
            the Comments section, below, for the values
            you can supply to this parameter.

            <- attrP  Pointer to a UInt32 into which the specified
            attribute value is placed by this function.

Result  Returns errNone if the function successfully retrieved the specified
        attribute, or sysErrParamErr if selector doesn’t represent a
        screen attribute.

Comments  This function returns many of the attributes that can be obtained
          with WinScreenMode. Unlike WinScreenMode, however, this
          function can also return the number of bytes used by each row in
          the screen buffer as well as the number of pixels per inch on the
          screen’s x and y axes.

          Unlike WinScreenMode, you cannot set any attributes with this
          function. Also, you cannot use this function to obtain the “color
          enabled” attribute. And unlike WinScreenMode, this function
          always returns the true screen dimensions; WinScreenMode
          converts the dimensions to the active coordinate system.
Applications can use the screen resolution information to make intelligent decisions about how to draw primitives on Palm Powered handhelds with different screen resolutions.

**WinScreenAttrType**

This enum defines the selectors that can be used with the `WinScreenGetAttribute` function.

```c
typedef enum {
    winScreenWidth,
    winScreenHeight,
    winScreenRowBytes,
    winScreenDepth,
    winScreenAllDepths,
    winScreenDensity,
    winScreenPixelFormat,
    winScreenResolutionX,
    winScreenResolutionY
} WinScreenAttrType;
```

**Value Descriptions**

- **winScreenWidth** The width of the screen, in pixels.
- **winScreenHeight** The height of the screen, in pixels.
- **winScreenRowBytes** The number of bytes used by each row in the screen buffer.
- **winScreenDepth** The screen depth.
- **winScreenAllDepths** All screen depths (in bitmap format).
- **winScreenDensity** The screen bitmap’s density.
- **winScreenPixelFormat** The `PixelFormatType` appropriate for the screen.
- **winScreenResolutionX** The number of pixels per inch along the screen’s x axis.
- **winScreenResolutionY** The number of pixels per inch along the screen’s y axis.

**Compatibility** Implemented only if the [High-Density Display Feature Set](#) is
WinScreenLock

**Purpose**
“Lock” the current screen by switching the UI concept of the screen base address to an area that is not reflected on the display.

**Declared In**
Window.h

**Prototype**
UInt8 *WinScreenLock (WinLockInitType initMode)

**Parameters**
-> initMode
Indicates how to initialize the new screen area. Specify one of the following values:

- winLockCopy
  Copy old screen to new.

- winLockErase
  Erase new screen to white.

- winLockDontCare
  Don't do anything

**Result**
Returns a pointer to the new screen base address, or NULL if this routine fails.

**Comments**
This routine can be used to “freeze” the display while doing lengthy drawing operations to avoid a flickering effect. Call WinScreenUnlock to unlock the display and cause it to be updated with any changes. The screen must be unlocked as many times as it is locked to actually update the display.

Because this function copies the screen, using it is a relatively expensive operation.

**Compatibility**
Implemented only if 3.5 New Feature Set is present.
WinScreenMode

Purpose
Sets or returns display parameters, including display geometry, bit depth, and color support.

Declared In
Window.h

Prototype
Err WinScreenMode
(WinScreenModeOperation operation,
UInt32 *widthP, UInt32 *heightP, UInt32 *depthP,
Boolean *enableColorP)

Parameters
The widthP, heightP, depthP, and enableColorP parameters are used in different ways for different operations. See Comments at the end of this description for details.

-> operation
The work this function is to perform, as specified by one of the following selectors:

winScreenModeGet
Return the current settings for the display.

winScreenModeGetDefaults
Return the default settings for the display.

winScreenModeGetSupportedDepths
Return in depthP a hexadecimal value indicating the supported screen depths. The binary representation of this value defines a bitfield in which the value 1 indicates support for a particular display depth. The position representing a particular bit depth corresponds to the value $2^{(bitDepth-1)}$. See the Example at the end of this function description for more information.
winScreenModeGetSupportsColor
Return true as the value of the enableColorP parameter when color mode can be enabled.

winScreenModeSet
Change display settings to the values specified by the other arguments to the WinScreenMode function.

winScreenModeSetToDefaults
Change display settings to default values.

<> widthP Pointer to new/old screen width. For backward compatibility, when operation is winScreenModeGet or winScreenModeGetDefaults, a single-density width is returned, even if the handheld has a double-density display. Use WinScreenGetAttribute to retrieve the true hardware dimensions of the display.

<> heightP Pointer to new/old screen height. For backward compatibility, when operation is winScreenModeGet or winScreenModeGetDefaults, a single-density height is returned, even if the handheld has a double-density display. Use WinScreenGetAttribute to retrieve the true hardware dimensions of the display.

<> depthP Pointer to new/old/available screen depth.
<-> enableColorP

Pass true to enable color drawing mode. The returned value (when using an operation that returns a value through this parameter) simply indicates whether or not the hardware supports color; its value does not change based on the current screen depth.

**Result**

If no error, returns values as specified by the operation argument. Various invalid arguments may cause this function to return a sysErrParamErr result code. In rare cases, a failed allocation can cause this function to return a memErrNotEnoughSpace error.

**Comments**

The widthP, heightP, depthP, and enableColorP parameters are used in different ways for different operations. All “get” operations overwrite these values with a result when the function returns. The winScreenModeSet operation changes current display parameters when passed valid argument values that are not NULL pointers. The winScreenModeSetToDefaults operation ignores values passed for all of these parameters.

*Table 54.1* summarizes parameter usage for each operation this function performs.
This function ignores NULL pointer arguments to the `widthP`, `heightP`, `depthP`, and `enableColorP` parameters; thus, you can pass a NULL pointer for any of these values to leave the current value unchanged. Similarly, when getting values, this function does not return a value for any NULL pointer argument.

If you change the display depth, it is recommended that you restore it to its previous state when your application closes, even though the system sets display parameters back to their default values when launching an application.

Note that none of the other operations interprets the depth parameter the same way that `winScreenModeGetSupportedDepths` does. For example, to set the display depth to 8-bit mode, you use 8 (decimal) for the display depth, not 0x80 (128 decimal).

When a window is created, and if the window's associated bitmap does not have its own color table, the window will use the system's default color translation tables when a blitting operation occurs to that window. When the system's bit depth changes, the system's default color translation tables are recalculated based on the new screen depth. When the blit occurs at the new screen depth to the offscreen window, the color translation tables are out of sync.

To workaround this system limitation, developers should either:

- allocate offscreen windows after changing the depth, or

---

**Table 54.1 Use of parameters to WinScreenMode function**

<table>
<thead>
<tr>
<th>Operation</th>
<th>widthP</th>
<th>heightP</th>
<th>depthP</th>
<th>enableColorP</th>
</tr>
</thead>
<tbody>
<tr>
<td>...Get</td>
<td>returned</td>
<td>returned</td>
<td>returned</td>
<td>returned</td>
</tr>
<tr>
<td>...GetDefaults</td>
<td>returned</td>
<td>returned</td>
<td>returned</td>
<td>returned</td>
</tr>
<tr>
<td>...GetSupportedDepths</td>
<td>ignored</td>
<td>ignored</td>
<td>returned</td>
<td>pass in</td>
</tr>
<tr>
<td>...GetSupportsColor</td>
<td>ignored</td>
<td>ignored</td>
<td>pass in</td>
<td>returned</td>
</tr>
<tr>
<td>...Set</td>
<td>pass in</td>
<td>pass in</td>
<td>pass in</td>
<td>pass in</td>
</tr>
<tr>
<td>...SetToDefaults</td>
<td>ignored</td>
<td>ignored</td>
<td>ignored</td>
<td>ignored</td>
</tr>
</tbody>
</table>
• use `WinCreateBitmapWindow` so that it uses a bitmap with a defined color table.

The latter workaround causes the system to perform color matching when blitting, so the first workaround may be preferred.

**Compatibility**

Implemented only if [3.5 New Feature Set](#) is present. In OS versions prior to 3.5, this function is called `ScrDisplayMode`. The prototype for `ScrDisplayMode` is similar to `WinScreenMode`:

```c
Err ScrDisplayMode (
    ScrDisplayModeOperation operation, 
    DWordPtr widthP, DWordPtr heightP, 
    DWordPtr depthP, BooleanPtr enableColorP)
```

The only other difference between `ScrDisplayMode` and `WinScreenMode` is that the `ScrDisplayModeOperation` constants begin with the prefix `scrDisplayMode` rather than `winScreenMode`.

**Example**

Here are some additional examples of return values provided by the `winScreenModeGetSupportedDepths` mode of the `WinScreenMode` function.

This function indicates support for 4-bit drawing by returning a value of `0x08`, or $2^3$, which corresponds to a binary value of $1000$. Support for bit depths of 2 and 1 is indicated by a return value of $0x03$. Support for bit depths of 4, 2, and 1 is indicated by $0x0B$, which is a binary value of $1011$. Support for bit depths of 16, 8, 4 and 2 is indicated by $0x808A$. The figure immediately following depicts this final example graphically.

![Diagram](#)

**See Also**

[WinScreenGetAttribute](#)
 WinScreenUnlock

Purpose Unlock the screen and update the display.

Declared In Window.h

Prototype void WinScreenUnlock (void)

Parameters None.

Result Returns nothing.

Comments The screen must be unlocked as many times as it is locked to actually update the display.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinScreenLock

WinScrollRectangle

Purpose Scroll a rectangle in the draw window.

Declared In Window.h

Prototype void WinScrollRectangle (const RectangleType *rP, WinDirectionType direction, Coord distance, RectangleType *vacatedP)

Parameters -> rP Pointer to the rectangle to scroll.
   -> direction Direction to scroll (winUp, winDown, winLeft, or winRight).
   -> distance Distance to scroll in pixels.
<- vacatedP  Pointer to the rectangle that needs to be
drawn because it has been vacated as a result
of the scroll.

Result  Returns nothing.

Comments  The rectangle scrolls within its own bounds. Any portion of the
rectangle that is scrolled outside its bounds is clipped.

WinSetActiveWindow

Purpose  Make a window the active window.

Declared In  Window.h

Prototype  void WinSetActiveWindow (WinHandle winHandle)

Parameters  -> winHandle  Handle of a window.

Result  Returns nothing.

Comments  The active window is not actually set in this routine; flags are set to
indicate that a window is being exited and another window is being
entered. The routine EvtGetEvent sends a winExitEvent and a
winEnterEvent when it detects these flags. The active window is
set by EvtGetEvent when it sends the winEnterEvent. The
draw window is also set to the new active window when the active
window is changed.

The window is enabled before it is made active.

All user input is directed to the active window.

See Also  WinGetActiveWindow, EvtGetEvent
**WinSetBackColor**

**Purpose**  
Set the background color to use in subsequent draw operations.

**Declared In**  
Window.h

**Prototype**  
IndexedColorType WinSetBackColor  
(IndexedColorType backColor)

**Parameters**  
-> backColor  
Color to set; specify a value of type IndexedColorType.

**Result**  
Returns the previous background color index.

**Comments**  
This function changes the current drawing state. If necessary, use WinPushDrawState to preserve the current drawing state before you set this function and use WinPopDrawState to restore it later.

To set the foreground color to a predefined UI color default, use UIColorGetTableEntryIndex as an input to this function. For example:

```
curColor = WinSetBackColor  
(UIColorGetTableEntryIndex(UIFieldBackground));
```

**Compatibility**  
Implemented only if 3.5 New Feature Set is present.

**See Also**  
WinSetForeColor, WinSetTextColor
**WinSetBackColorRGB**

**Purpose**
Set the background color to use in subsequent draw operations.

**Declared In**
Window.h

**Prototype**
```c
void WinSetBackColorRGB
   (const RGBColorType *newRgbP,
    RGBColorType *prevRgbP)
```

**Parameters**
- `newRgbP` Color to set; specify a value of type RGBColorType.
- `prevRgbP` Previous color; specify a value of type RGBColorType.

**Result**
Returns nothing

**Comments**
This function takes new and previous RGBColorType arguments. It is okay to set `newRgbP` or `prevRgbP` to NULL. If an application only wants to get the current color, the `newRgbP` argument is set to NULL. If the application does not care about the previous color, `prevRgbP` can be set to NULL.

This function sets the `backColorRGB` field of the DrawStateType structure to the value specified by `newRgbP`. It then sets the index field of `backColorRGB` to the 8 bit system palette entry that most closely matches the RGB components. Finally, it sets the `backColor` index field of DrawStateType to this index value.

This function changes the current drawing state. If necessary, use WinPushDrawState to preserve the current drawing state before you set this function and use WinPopDrawState to restore it later.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
WinSetForeColorRGB, WinSetTextColorRGB
WinSetBounds

Purpose
Set the bounds of the window to display-relative coordinates.

Declared In
Window.h

Prototype
void WinSetBounds (WinHandle winHandle, const RectangleType *rP)

Parameters
- `winHandle` Handle for the window for which to set the bounds.
- `rP` Pointer to a rectangle to use for bounds.

Result
Returns nothing.

Comments
A visible window cannot have its bounds modified.

Compatibility
Implemented only if 2.0 New Feature Set is present.

See Also
WinGetBounds

WinSetClip

Purpose
Set the clipping rectangle of the draw window.

Declared In
Window.h

Prototype
void WinSetClip (const RectangleType *rP)

Parameters
- `rP` Pointer to a structure holding the clipping bounds.

Result
Returns nothing.

See Also
WinClipRectangle, WinSetClip, WinGetClip
**WinSetDrawMode**

**Purpose**
Set the transfer mode to use in subsequent draw operations.

**Declared In**
Window.h

**Prototype**
```c
WinDrawOperation WinSetDrawMode
    (WinDrawOperation newMode)
```

**Parameters**
- `newMode` Transfer mode to set; specify one of the `WinDrawOperation` values.

**Result**
Returns the previous transfer mode.

**Comments**
This function changes the current drawing state. If necessary, use `WinPushDrawState` to preserve the current drawing state before you set this function and use `WinPopDrawState` to restore it later.

**Compatibility**
Implemented only if 3.5 New Feature Set is present.

**New**
**WinSetCoordinateSystem**

**Purpose**
Establish the coordinate system to be used for subsequent drawing operations.

**Declared In**
Window.h

**Prototype**
```c
UInt16 WinSetCoordinateSystem (UInt16 coordSys)
```

**Parameters**
- `coordSys` The desired coordinate system. Supply one of the values defined in “Window Coordinate System Constants” on page 1163.

**Result**
Returns the previous coordinate system value.
Comments  
This function modifies the `scale` field in the draw state (a `DrawStateType` structure). As when making other modifications to a window’s draw state, applications should call `WinPushDrawState` before modifying the coordinate system. To restore the coordinate system, your application can then call `WinPopDrawState`.

To calculate the draw state `scale` field, the Window Manager divides the density of the bitmap associated with the draw window by `coordSys`. If `coordSys` is `kCoordinatesNative`, the Window Manager sets the scale field to 1.0, which enables 1-to-1 mapping of coordinates to pixels.

If you supply a value of `kCoordinatesStandard` for `coordSys`, subsequent drawing will use the standard coordinate system.

Compatibility  
Implemented only if the `High-Density Display Feature Set` is present.

See Also  
`WinGetCoordinateSystem`  
`WinSetDrawWindow`

**WinSetDrawWindow**

Purpose  
Set the draw window. (All drawing operations are relative to the draw window.)

Declared In  
`Window.h`

Prototype  
`WinHandle WinSetDrawWindow (WinHandle winHandle)`

Parameters  
`-> winHandle`  
Handle of a window.

Result  
Returns the previous draw window.

Compatibility  
OS versions before 3.5 allowed you to use `NULL` as a parameter to this function to set the draw window to the display window (or screen window). In version 3.5 and higher, this practice is discouraged. If `winHandle` is `NULL`, the debug ROM sets the draw
window to badDrawWindowValue, and you are warned if you try to draw to it.

See Also  WinGetDrawWindow, WinSetActiveWindow

WinSetForeColor

Purpose  Set the foreground color to use in subsequent draw operations.

Declared In  Window.h

Prototype  IndexedColorType WinSetForeColor

Parameters  -> foreColor  Color to set; specify a value of type IndexedColorType.

Result  Returns the previous foreground color index.

Comments  This function changes the current drawing state. If necessary, use WinPushDrawState to preserve the current drawing state before you set this function and use WinPopDrawState to restore it later.

To set the foreground color to a predefined UI color default, use UIColorGetTableEntryIndex as an input to this function. For example:

```c
curColor = WinSetForeColor
    (UIColorGetTableEntryIndex
    (UIObjectForeground));
```

Compatibility  Implemented only if 3.5 New Feature Set is present.

See Also  WinSetBackColor, WinSetTextColor
WinSetForeColorRGB

Purpose Set the foreground color to use in subsequent draw operations.

Declared In Window.h

Prototype

void WinSetForeColorRGB
(const RGBColorType *newRgbP,
RGBColorType *prevRgbP)

Parameters

-> newRgbP Color to set; specify a value of type RGBColorType.

<- prevRgbP Previous color; specify a value of type RGBColorType.

Result Returns nothing.

Comments This function takes new and previous RGBColorType arguments. It is okay to set newRgbP or prevRgbP to NULL. If an application only wants to get the current color, the newRgbP argument is set to NULL. If the application does not care about the previous color, prevRgbP can be set to NULL.

This function sets the foreColorRGB field of the DrawStateType structure to the value specified by newRgbP. It then sets the index field of foreColorRGB to the 8 bit system palette entry that most closely matches the RGB components. Finally, it sets the foreColor index field of DrawStateType to this index value.

This function changes the current drawing state. If necessary, use WinPushDrawState to preserve the current drawing state before you set this function and use WinPopDrawState to restore it later.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also WinSetBackColorRGB, WinSetTextColorRGB
**WinSetPattern**

**Purpose**
Set the current fill pattern.

**Declared In**
Window.h

**Prototype**
```c
void WinSetPattern
(const CustomPatternType *patternP)
```

**Parameters**
- `patternP`  
  Pattern to set (see `CustomPatternType`).

**Result**
Returns nothing.

**Comments**
The fill pattern is used by `WinFillLine` and `WinFillRectangle`. This function changes the current drawing state. If necessary, use `WinPushDrawState` to preserve the current drawing state before you set this function and use `WinPopDrawState` to restore it later.

**See Also**
`WinGetPattern`

---

**WinSetPatternType**

**Purpose**
Set the current pattern type.

**Declared In**
Window.h

**Prototype**
```c
void WinSetPatternType (PatternType newPattern)
```

**Parameters**
- `newPattern`  
  Pattern type to set for the draw window (see `PatternType`).

**Result**
Returns nothing.

**Comments**
This function sets the `pattern` field of the drawing state to `newPattern` and sets the `patternData` field to `NULL`. To set `patternData` to a custom pattern use `WinSetPattern`.

The fill pattern is used by `WinFillLine` and `WinFillRectangle`.  

---

1238  Palm OS Programmer’s API Reference
This function changes the current drawing state. If necessary, use \texttt{WinPushDrawState} to preserve the current drawing state before you set this function and use \texttt{WinPopDrawState} to restore it later.

**Compatibility**
Implemented only if \texttt{3.5 New Feature Set} is present.

**See Also**
\texttt{WinGetPatternType}

\textbf{WinSetTextColor}

**Purpose**
Set the color to use for drawing characters in subsequent draw operations.

**Declared In**
Window.h

**Prototype**
\texttt{IndexedColorType WinSetTextColor (IndexedColorType textColor)}

**Parameters**
-> textColor Color to set; specify a value of type \texttt{IndexedColorType}.

**Result**
Returns the previous text color index.

**Comments**
This function changes the current drawing state. If necessary, use \texttt{WinPushDrawState} to preserve the current drawing state before you set this function and use \texttt{WinPopDrawState} to restore it later.

To set the foreground color to a predefined UI color default, use \texttt{UIColorGetTableEntryIndex} as an input to this function. For example:

\[
\text{curColor} = \text{WinSetTextColor} (\text{UIColorGetTableEntryIndex} (\text{UIFieldText}));
\]

**Compatibility**
Implemented only if \texttt{3.5 New Feature Set} is present.

**See Also**
\texttt{WinSetBackColor, WinSetForeColor}
WinSetTextColorRGB

Purpose Set the color to use for drawing characters in subsequent draw operations.

Declared In Window.h

Prototype

```c
void WinSetTextColorRGB
(const RGBColorType *newRgbP,
RGBColorType *prevRgbP)
```

Parameters

- **newRgbP** Color to set; specify a value of type `RGBColorType`.
- **prevRgbP** Previous color; specify a value of type `RGBColorType`.

Result Returns nothing.

Comments

This function takes new and previous `RGBColorType` arguments. It is okay to set `newRgbP` or `prevRgbP` to NULL. If an application only wants to get the current color, the `newRgbP` argument is set to NULL. If the application does not care about the previous color, `prevRgbP` can be set to NULL.

This function sets the `textColorRGB` field of the `DrawStateType` structure to the value specified by `newRgbP`. It then sets the index field of `textColorRGB` to the 8 bit system palette entry that most closely matches the RGB components. Finally, it sets the `textColor` index field of `DrawStateType` to this index value.

This function changes the current drawing state. If necessary, use `WinPushDrawState` to preserve the current drawing state before you set this function and use `WinPopDrawState` to restore it later.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `WinSetBackColorRGB`, `WinSetForeColorRGB`
**WinSetUnderlineMode**

**Purpose**
Set the graphic state to enable or disable the underlining of characters.

**Declared In**
Window.h

**Prototype**
```
UnderlineModeType WinSetUnderlineMode
(UnderlineModeType mode)
```

**Parameters**
- `mode`: New underline mode type; see `UnderlineModeType`.

**Result**
Returns the previous underline mode type.

**Comments**
This function changes the current drawing state. If necessary, use `WinPushDrawState` to preserve the current drawing state before you set this function and use `WinPopDrawState` to restore it later.

**See Also**
- WinDrawChars

---

**New WinUnscaleCoord**

**Purpose**
Convert a single coordinate from the active coordinate system to the standard coordinate system.

**Declared In**
Window.h

**Prototype**
```
Coord WinUnscaleCoord
(Coord coord,
Boolean ceiling)
```

**Parameters**
- `coord`: A coordinate in the active coordinate system.
- `ceiling`: Pass `true` to round up, `false` to truncate the fractional part when scaling.

**Result**
Returns the coordinate scaled to the standard coordinate system.
Comments

This function converts a coordinate by dividing it by the coordinate scaling factor, truncating or rounding the result to an integer value depending on the value of `ceiling`.

If the active coordinate system is `kCoordinatesStandard`, the returned coordinate is equal to the supplied coordinate.

Compatibility

Implemented only if the High-Density Display Feature Set is present.

See Also

WinScaleCoord, WinUnscalePoint, WinUnscaleRectangle

New WinUnscalePoint

Purpose

Convert a point from the active coordinate system to the standard coordinate system.

Declared In

Window.h

Prototype

void WinUnscalePoint (PointType *pointP, Boolean ceiling)

Parameters

<-> pointP   Pointer to a PointType structure that, before the call, should contain a point's coordinates using the active coordinate system. After this function is called the PointType structure contains the coordinates of the point scaled to the standard coordinate system.

-> ceiling   Pass true to round up, false to truncate the fractional part when scaling.

Result

Returns nothing. The coordinates of the point indicated by `pointP` are converted to the standard coordinate system.

Comments

This function converts a point by dividing its x and y coordinates by the coordinate scaling factor, truncating or rounding the results to integer values depending on the value of `ceiling`. For instance,
the input coordinates (11, 13) are transformed to (6, 7) if the input values represent native coordinates on a handheld with a double-density screen and ceiling is set to true. If ceiling is set to false, the same input coordinates are transformed to (5, 6).

If the active coordinate system is kCoordinatesStandard, pointP is not changed by this function.

Compatibility  Implemented only if the High-Density Display Feature Set is present.

See Also  WinScalePoint, WinUnscaleCoord, WinUnscaleRectangle

New  WinUnscaleRectangle

Purpose  Convert a rectangle from the active coordinate system to the standard coordinate system.

Declared In  Window.h

Prototype  void WinUnscaleRectangle (RectangleType *rectP)

Parameters  

Result  Returns nothing. The coordinates of the rectangle indicated by rectP are converted to the standard coordinate system.

Comments  This function scales the rectangle’s topLeft and extent points by dividing their x and y coordinates by the coordinate scaling factor. All values are then truncated, but if either topLeft.x or extent.x had a fractional part, extent.x is incremented by 1.
(and, similarly, if either topLeft.y or extent.y had a fractional part, extent.y is incremented by 1).
If the active coordinate system is kCoordinatesStandard, rectP is not changed by this function.

**Compatibility**
Implemented only if the [High-Density Display Feature Set](#) is present.

**See Also**
[WinScaleRectangle](#), [WinUnscaleCoord](#), [WinUnscalePoint](#)

### WinValidateHandle

**Purpose**
Return true if the specified handle references a valid window object.

**Declared In**
Window.h

**Prototype**
```c
Boolean WinValidateHandle (WinHandle winHandle)
```

**Parameters**
- `-> winHandle` The handle to be tested.

**Result**
Returns true if the specified handle references a non-NULL pointer to a window in the active window list, false if the handle references a window whose values are out of sync with the current system state.

**Comments**
For debugging purposes only. Do not include this function in commercial applications.

**Compatibility**
Implemented only if [3.0 New Feature Set](#) is present.

**See Also**
[FrmValidatePtr](#), [FrmRemoveObject](#)
**WinWindowToDisplayPt**

**Purpose**  
Convert a window-relative coordinate to a display-relative coordinate.

**Declared In**  
Window.h

**Prototype**  
void WinWindowToDisplayPt (Coord *extentX, Coord *extentY)

**Parameters**  
<-> extentX  
Pointer to x coordinate to convert.  
<-> extentY  
Pointer to y coordinate to convert.

**Result**  
Returns nothing.

**Comments**  
The coordinate passed is assumed to be relative to the draw window.

**See Also**  
WinDisplayToWindowPt
This chapter describes miscellaneous system functions. The functions in this chapter are declared in the header files Crc.h, DLServer.h, IntlMgr.h, and Localize.h.

**Crc16CalcBlock**

**Purpose** Calculate the 16-bit CRC of a data block using the table lookup method.

**Declared In** Crc.h

**Prototype**

```
UInt16 Crc16CalcBlock (const void *bufP, UInt16 count, UInt16 crc)
```

**Parameters**

- `bufP` Pointer to the data buffer.
- `count` Number of bytes in the buffer.
- `crc` Seed crc value.

**Result** A 16-bit CRC for the data buffer.
DlkControl

Purpose
Perform an operation at the behest of the desktop software. Among other things, this function is used to return values to the conduit during the handling of `sysAppLaunchCmdHandleSyncCallApp`.

Declared In
DLServer.h

Prototype
```c
Err DlkControl (DlkCtlEnum op, void *param1P, void *param2P)
```

Parameters
- `-> op` Desktop Link control code. Use `dlkCtlSendCallAppReply` when sending a result back to the conduit while handling a `sysAppLaunchCmdHandleSyncCallApp` launch code.
- `<-> param1P` Pointer to the first parameter (operation-specific). For `dlkCtlSendCallAppReply`, this parameter should point to a `DlkCallAppReplyParamType` structure.
- `<-> param2P` Pointer to the second parameter (operation-specific). For `dlkCtlSendCallAppReply`, this parameter should be set to NULL.

Result
`errNone` if no error, or an error code if there was a problem during the call to `DlkControl`. In either case, place the value returned from `DlkControl` into the `replyErr` field of the `SysAppLaunchCmdHandleSyncCallAppType` structure when handling `sysAppLaunchCmdHandleSyncCallApp`.

Comments
This function is needed to return data back to a conduit during the handling of `sysAppLaunchCmdHandleSyncCallApp`. Set `param1P` to point to a `DlkCallAppReplyParamType` structure, as described below. See the Example on page 1249 for an illustration of how to handle `sysAppLaunchCmdHandleSyncCallApp`. 
**DlkCallAppReplyParamType**

**Prototype**

typedef struct DlkCallAppReplyParamType {
    UInt16 pbSize;
    UInt32 dwResultCode;
    const void *resultP;
    UInt32 dwResultSize;
    void *dlRefP;
    UInt32 dwReserved1;
} DlkCallAppReplyParamType;

**Fields**

- **pbSize**: Size of this parameter block. Set it to `sizeof(DlkCallAppReplyParamType)`.
- **dwResultCode**: Result code to be returned to the remote caller.
- **resultP**: Pointer to result data.
- **dwResultSize**: Size of result data block (number of bytes).
- **dlRefP**: Desktop Link reference pointer from `SysAppLaunchCmdHandleSyncCallAppType`.
- **dwReserved1**: Reserved. Set to `NULL`.

**Example**

The `SysAppLaunchCmdHandleSyncCallAppType` structure that accompanies the `sysAppLaunchCmdHandleSyncCallApp` launch code contains all of the information passed into `SyncCallRemoteModule` on the desktop as well as the necessary fields to pass the result pack to the desktop. At the end of your `sysAppLaunchCmdHandleSyncCallApp` launch code handler, you'll need to send a `DlkCallAppReplyParamType` reply structure back to the device using `DlkControl`.

```c
#include <DLServer.h>
...
case sysAppLaunchCmdHandleSyncCallApp:
{
    SysAppLaunchCmdHandleSyncCallAppType *theCommandPtr;
    DlkCallAppReplyParamType theReplyParams;
    CharPtr theReplyBuffer = "SUCCESS";

    // Cast the cmdPBP to a SysAppLaunchCmdHandleSyncCallAppType
    // pointer so that we can work with it.
    theCommandPtr = (SysAppLaunchCmdHandleSyncCallAppType*)cmdPBP;
```
// Do whatever work is necessary here. If you set the m_wActionCode
// field in your CCallModuleParams class on the desktop, then you
// can handle that code by looking at the action field of theCommandPtr
// (i.e.) if (theCommandPtr->action == 1)

// Create the reply to send back to the desktop

// First clear out all the fields. This is necessary so that the reserved
// fields are set to NULL.
MemSet( &theReplyParams, sizeof(DlkCallAppReplyParamType), 0 );

// Set the size of the reply. This is required.
theReplyParams.pbSize = sizeof(DlkCallAppReplyParamType);

// Set the result code. Normally this will be set to zero unless you want
// to send an error code back to the desktop.
theReplyParams.dwResultCode = 0;

// Fill in the reply buffer and buffer length
theReplyParams.resultP = theReplyBuffer;
theReplyParams.dwResultSize = StrLen(theReplyBuffer) + 1;

// Fill in the DL reference pointer. This is required.
theReplyParams.dlRefP = theCommandPtr->dlRefP;

// Set the handled field to true. This is required to let the desktop
// know that the sysAppLaunchCmdHandleSyncCallApp was handled. If you
// don't set this to true, the call to SyncCallRemoteModule will return
// SYNCERR_UNKNOWN_REQUEST.
theCommandPtr->handled = true;

// Finally, set the replyErr field by passing the reply parameters to
// DlkControl. This is required for the DLServer to properly handle the
// reply request.
theCommandPtr->replyErr = DlkControl (dlkCtlSendCallAppReply,
                   &theReplyParams, NULL);

break;

Table 55.1 and Table 55.2 list some important mappings from the
CCallModuleParams class on the desktop to the
SysAppLaunchCmdHandleSyncCallAppType and
DlkCallAppReplyParamType structures on the handheld.
Palm OS Programmer’s API Reference

Miscellaneous System Functions

DlkGetSyncInfo

Purpose
Get the sync info managed by Desktop Link. This function is often used to obtain the user name on the handheld.

Declared In
DLServer.h

Prototype
Err DlkGetSyncInfo (UInt32 *succSyncDateP,
UInt32 *lastSyncDateP,
DlkSyncStateType *syncStateP, Char *nameBufP,
Char *logBufP, Int32 *logLenP)

Parameters
- succSyncDateP
  Pointer to the location where the date of the last successful sync is stored. Supply NULL for this parameter if this date isn’t needed.
<- lastSyncDateP  Pointer to the location where the date of the last sync, successful or otherwise, is stored. Supply NULL for this parameter if this date isn’t needed.

<- syncStateP  Pointer to a DlkSyncStateType enum into which the state of the last sync is stored. Supply NULL for this parameter if the state information isn’t needed. See the Comments, below, for a description of this enum.

<- nameBufP  Pointer to a string buffer into which the null-terminated handheld user name is stored. This string buffer must have been preallocated to be at least dlkUserNameBufSize bytes in length. Supply NULL for this parameter if the user name isn’t needed.

<- logBufP  Pointer to a string buffer into which the sync log text, null-terminated, is stored. Supply NULL for this parameter if the log text isn’t needed. If you supply a valid pointer for this parameter, you must specify the preallocated buffer length using the logLenP parameter; the returned log text will be truncated, if necessary, to fit within the buffer.

<- logLenP  Pointer to the log buffer size. If logBufP is not NULL, on entry you must set this value to the size of the logBufP buffer. When this function returns, this value indicates the actual length of the log text, not counting the null terminator.

Result  Returns errNone if no error, or dlkErrMemory if the Desktop Link preferences resource couldn’t be locked.

Comments  The state information returned through syncStateP has one of the values defined by the DlkSyncStateType enum:

```c
typedef enum DlkSyncStateType {
    dlkSyncStateNeverSynced = 0,
    dlkSyncStateInProgress,
```
dlkSyncStateLostConnection,
dlkSyncStateLocalCan,
dlkSyncStateRemoteCan,
dlkSyncStateLowMemoryOnTD,
dlkSyncStateAborted,
dlkSyncStateCompleted,
dlkSyncStateIncompatibleProducts,
dlkSyncStateNPOD
} DlkSyncStateType;

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dlkSyncStateNeverSynced</td>
<td>The handheld has never been synced.</td>
</tr>
<tr>
<td>dlkSyncStateInProgress</td>
<td>A sync is currently in progress.</td>
</tr>
<tr>
<td>dlkSyncStateLostConnection</td>
<td>The connection was lost during sync.</td>
</tr>
<tr>
<td>dlkSyncStateLocalCan</td>
<td>Sync was cancelled by the user on the handheld.</td>
</tr>
<tr>
<td>dlkSyncStateRemoteCan</td>
<td>Sync was cancelled by the user from the desktop.</td>
</tr>
<tr>
<td>dlkSyncStateLowMemoryOnTD</td>
<td>Sync ended due to a low memory condition on the handheld.</td>
</tr>
<tr>
<td>dlkSyncStateAborted</td>
<td>Sync was aborted for some other reason.</td>
</tr>
<tr>
<td>dlkSyncStateCompleted</td>
<td>Sync completed normally.</td>
</tr>
</tbody>
</table>
**Example**

This function is most often used to obtain the handheld user name. The following code excerpt shows how to do this (for clarity, error-checking has been omitted):

```c
MemHandle nameH;
char *nameP;

// Allocate a buffer for the user name
nameH = MemHandleNew(dlkUserNameBufSize);
nameP = MemHandleLock(nameH);

// Obtain the user's name
DlkGetSyncInfo(NULL, NULL, NULL, nameP, NULL, NULL);

// ... Do something with the user name here ...

// Now that we're done with the user name, free the buffer
MemPtrUnlock(nameP);
```

**Compatibility**

The `dlkSyncStateIncompatibleProducts` enum value was added in Palm OS 3.0. The `dlkSyncStateNPOD` enum value was added in Palm OS 4.0.
### IntlGetRoutineAddress

**Purpose**
Return the address of an international manager or text manager function.

**Declared In**
IntlMgr.h

**Prototype**
```c
void *IntlGetRoutineAddress(IntlSelector inSelector)
```

**Parameters**
- `inSelector`
  One of the routine selectors defined in IntlMgr.h.

**Result**
Returns the address of the corresponding function. Returns NULL if an invalid routine selector is passed.

**Comments**
Use this function for performance reasons. It returns the address of an international manager or text manager function. You can then use this address to call the function without having to go through the international manager’s trap dispatch table. This function is mostly useful for optimizing the performance of text manager routines that are called in a tight loop.

You might also use this function to check for the presence of newer international manager and text manager functions. If the result is NULL, the function is not implemented on this device.

**Compatibility**
Implemented only if International Feature Set is present.

**See Also**
IntlSetRoutineAddress, SysGetTrapAddress
IntlSetRoutineAddress

**Purpose**
Set the address of the function corresponding to an international manager or text manager function.

**Declared In**
IntlMgr.h

**Prototype**
```
Err IntlSetRoutineAddress
    (IntlSelector iSelector, void *iProcPtr)
```

**Parameters**
- `iSelector` One of the routine selectors defined in IntlMgr.h.
- `iProcPtr` Pointer to a function that the routine identified by `iSelector` should point to.

**Result**
Returns `errNone` if no error, or `intlErrInvalidSelector` if `iSelector` does not refer to a valid international manager or text manager routine.

**Comments**
This function is useful for patching an international or text manager function. Normally only a locale module would need to patch one of these functions.

**WARNING!**
If your application patches an international manager function using this function, you **must** remove the patch before your application exits. Do **not** use this mechanism to permanently patch international manager functions as it may cause unpredictable results for the system and other applications.

**Compatibility**
Implemented only if [4.0 New Feature Set](#) is present. If [5.0 New Feature Set](#) is present this function is unimplemented.

**See Also**
[IntlGetRoutineAddress](#), [SysSetTrapAddress](#)
LocGetNumberSeparators

**Purpose**
Get localized number separators.

**Declared In**
Localize.h

**Prototype**
```c
void LocGetNumberSeparators
(NumberFormatType numberFormat,
Char *thousandSeparator, Char *decimalSeparator)
```

**Parameters**
- `numberFormat` The format to use (see `NumberFormatType`).
- `thousandSeparator` The character used for the thousands separator.
- `decimalSeparator` The character used for the decimal separator.

**Result**
Returns nothing.

**Comments**
The format to use is stored in the system preferences. You can obtain it by passing `prefNumberFormat` to `PrefGetPreference`.

**Compatibility**
Implemented only if 2.0 New Feature Set is present.

**See Also**
`StrLocalizeNumber`, `StrDelocalizeNumber`, “Localized Applications” in the Palm OS Programmer’s Companion, vol. I
New

**PceNativeCall**

**Purpose**
Call a native ARM or Windows NT function from code running in the PACE (68k) environment.

**Declared In**
PceNativeCall.h

**Prototype**

```c
UInt32 PceNativeCall(
    NativeFuncType *nativeFuncP, void *userDataP)
```

**Parameters**

- `nativeFuncP` On a handheld with an ARM processor, this is a pointer to the ARM function to be executed. On Palm OS Simulator, this is a pointer to the name of a DLL and the name of the entry point within that DLL that is to be executed, separated by a null character and terminated with a null character. See the Comments, below, for more details on the format of this argument.

- `userDataP` Pointer to an application-specific block of data that is passed to the ARM function. This block has no specific alignment requirements; it needn’t be aligned on a 16- or 32-bit boundary. Note that your ARM function may impose specific alignment requirements, however.

**Result**
The return value of the specified ARM function is returned by PceNativeCall. This value is placed in both the A0 and D0 registers in the emulated 68k CPU, allowing PceNativeCall to support both pointer and immediate return value conventions.

**Comments**
Applications that employ PceNativeCall won’t work on handhelds running a version of Palm OS prior to Palm OS 5. Before calling PceNativeCall, your application must verify the underlying processor type, since the calling convention is different on Palm OS Simulator. See “Calling an ARM Function” on page 340 of Palm OS Programmer’s Companion, vol. I for more information and an example.
PceNativeCall byte-swaps the parameter pointer and return value as appropriate for the Dragonball-to-ARM transition. This allows you to dereference userDataP directly from your ARM code. Because the operating system has no knowledge of the structure of the parameter block, however, it performs no byte-swapping within this block. Your ARM code must do this as necessary for your application (see “Accessing 68K Data From an ARM Function” on page 342 of the Palm OS Programmer’s Companion, vol. I for more information).

On Palm OS Simulator, rather than passing a pointer to a block of ARM code in nativeFuncP, you instead pass a pointer to the name of a DLL and the name of the function within that DLL that is to be executed. These two names must be separated by a null character, and the entire sequence must be terminated by a null character. For example, to load the DLL found at C:\TEST_DLL\Debug\Simple.dll and call the function TestNativeCall within that DLL, you might pass a pointer to the following character string literal:

"C:\\TEST_DLL\\Debug\\Simple.dll\\0TestNativeCall"

Note that if you don’t supply an absolute path, Simulator looks for the DLL in (or relative to) the directory from which PalmSim.exe is running. Thus, if the DLL is located in the same directory as PalmSim.exe, you can call the above function with:

"Simple.dll\\0TestNativeCall"

On release ROMs, PceNativeCall fails silently if nativeFuncP is NULL. On debug ROMs, it generates an error. All other pointers are treated as valid code and followed. If nativeFuncP is invalid, the processor will try to execute the code anyway and will eventually generate an error.

NOTE: The call to your native function is guaranteed to be made from ARM mode.

For more information on how the ARM code should be structured and how to call back and forth between the PACE and ARM environments, see “ARM-Native Functions” on page 339 of the Palm OS Programmer’s Companion, vol. I.
Compatibility Implemented only if 5.0 New Feature Set is present.
Part III: Communications
Connection Manager

The Connection Manager allows other applications to access, add, and delete connection profiles contained in the Connection panel.

This chapter provides reference material for the Connection Manager API declared in the header file ConnectionMgr.h:

- Connection Manager Constants
- Connection Manager Functions

For more information on the Connection Manager, see the chapter “Serial Communication” on page 89 in the Palm OS Programmer’s Companion, vol. II, Communications.

Connection Manager Data Types

CncProfileID

The CncProfileID type uniquely identifies a connection profile within the Connection Manager profile database. You pass this ID as a parameter to several of the Connection Manager functions. You can obtain a connection’s profile ID using CncProfileGetIDFromName or CncProfileGetIDFromIndex.

typedef UInt32 CncProfileID

Compatibility Defined only if Connection Manager Feature Set is present.

Connection Manager Constants

Profile Parameter Constants

The Connection Manager defines the following constants to represent individual parameters in the preinstalled connection profiles. Not all parameters work for all types of profiles.
## Connection Manager

### Connection Manager Constants

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kCncParamBaud</td>
<td>UInt32</td>
<td>The baud rate to use for the connection.</td>
</tr>
<tr>
<td>kCncParamBluetoothDeviceAddr</td>
<td>buffer</td>
<td>The 48-bit address (BD_ADDR) of the device that the handheld is connected to through the Bluetooth port. This parameter is only valid if kCncParamPort specifies the Bluetooth port.</td>
</tr>
<tr>
<td>kCncParamBluetoothDeviceName</td>
<td>string</td>
<td>The name of the device that the handheld is connected to through the Bluetooth port. The device name uses the UTF-8 character encoding. This parameter is only valid if kCncParamPort specifies the Bluetooth port.</td>
</tr>
<tr>
<td>kCncParamCountryIndex</td>
<td>UInt16</td>
<td>The index into the list of strings returned by kCncParamIntlModemCountryStringList and kCncParamIntlModemResetStringList that provides the name of the country and the reset commands for this profile.</td>
</tr>
<tr>
<td>kCncParamDeviceKind</td>
<td>UInt16</td>
<td>The type of connection being made (general serial connection, connection to a modem, connection to a phone, and so on). This value is one of the Device Kind Constants.</td>
</tr>
<tr>
<td>kCncParamDialingMode</td>
<td>UInt8</td>
<td>For modem profiles, the dialing mode. 1 for Pulse dialing, 0 for TouchTone.</td>
</tr>
</tbody>
</table>
## Connection Manager Constants

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kCncParamCountryIndex</td>
<td>UInt16</td>
<td>The index into the list of strings returned by kCncParamIntlModemCountryStringList and kCncParamIntlModemResetStringList that provides the name of the country and the reset commands for this profile.</td>
</tr>
<tr>
<td>kCncParamDeviceKind</td>
<td>UInt16</td>
<td>The type of connection being made (general serial connection, connection to a modem, connection to a phone, and so on). This value is one of the Device Kind Constants.</td>
</tr>
<tr>
<td>kCncParamDialingMode</td>
<td>UInt8</td>
<td>For modem profiles, the dialing mode. 1 for Pulse dialing, 0 for TouchTone.</td>
</tr>
</tbody>
</table>
### Connection Manager

#### Connection Manager Constants

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kCncParamIntlModem</td>
<td>buffer</td>
<td>For modem profiles, a data buffer containing a list of possible countries. This list contains the countries in which the modem can operate and is shown in the Connection panel Details form for this profile. The selected country controls the dialing prefix and the modem reset string. The kCncParamCountryIndex parameter specifies the country selected from this list. The data buffer contains a packed block of null-terminated strings, each containing a country name. The first 16 bits of the data buffer (a UInt16 value) tells how many strings are contained in the block. You can use <code>SysFormPointerArrayToString</code> to convert the data buffer into an array of strings. Because this parameter value has a variable size, you must first call <code>CncProfileSettingGet</code> with a NULL data pointer to obtain the correct size.</td>
</tr>
<tr>
<td>CountryStringList</td>
<td>buffer</td>
<td>For modem profiles, a data buffer containing possible reset strings. The kCncParamCountryIndex parameter specifies which reset string from this list is to be used. The actual string used is itself stored in the kCncParamResetString parameter.</td>
</tr>
</tbody>
</table>

[kCncParamIntlModem](#)
The data buffer contains a packed block of null-terminated strings, each containing a reset string. The first 16 bits of the data buffer (a UInt16 value) tells how many strings are contained in the block. You can use SysFormPointerArrayToStrings to convert the data buffer into an array of strings.

Because this parameter value has a variable size, you must first call CncProfileSettingGet with a NULL data pointer to obtain the correct size.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kCncParamInvisible</td>
<td>system flag</td>
<td>If true, the profile is hidden from the user. If false, the profile is visible. This parameter is not currently used.</td>
</tr>
<tr>
<td>kCncParamLocked</td>
<td>system flag</td>
<td>If true, the profile is locked so that the user cannot edit it. If false, the profile can be edited. This parameter can be set by profiles, such as phone profiles, created by third party utilities.</td>
</tr>
<tr>
<td>kCncParamName</td>
<td>string</td>
<td>The name of the profile.</td>
</tr>
<tr>
<td>kCncParamNoDetails</td>
<td>system flag</td>
<td>If true, the profile details should not be displayed. If false, they can be displayed. The profile details are the parameters that appear in the Details form of the Connection panel.</td>
</tr>
</tbody>
</table>
### Connection Manager

#### Connection Manager Constants

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kCncParamNonEditable</td>
<td>system flag</td>
<td>If true, the Connection panel’s Edit form should be suppressed for this profile. If false, the Edit form can be displayed. This parameter differs from kCncParamLocked and kCncParamReadOnly in that it causes an alert to be displayed if the user taps the Edit button. The other parameters allow the Edit form to be displayed but do not allow changes to be made. Also, a non-editable profile can be deleted, but a read-only or locked profile cannot.</td>
</tr>
<tr>
<td>kCncParamPort</td>
<td>UInt32</td>
<td>The logical, physical, or virtual port identifier. See “Port Constants” on page 1557 in the “Serial Manager” chapter for more information.</td>
</tr>
<tr>
<td>kCncParam_PSDCreator</td>
<td>UInt32</td>
<td>For phone profiles, the creator ID of the phone driver.</td>
</tr>
<tr>
<td>kCncParam_PSDName</td>
<td>string</td>
<td>For phone profiles, the name of the phone driver.</td>
</tr>
<tr>
<td>kCncParam_PSDParameter Buffer</td>
<td>buffer</td>
<td>For phone profiles, a data buffer containing any necessary data that the phone driver needs to store. This parameter typically holds data that is set using the Details form of the Connection panel.</td>
</tr>
<tr>
<td>kCncParam_PSDType</td>
<td>UInt32</td>
<td>For phone profiles, the database type for the phone driver.</td>
</tr>
</tbody>
</table>
### Parameter Name | Parameter Type | Description
--- | --- | ---
`kCncParamReadOnly` | system flag | If true, the profile is read-only and cannot be edited. If false, the profile can be edited. This parameter is only intended to be used by profiles pre-installed in the Palm OS.

`kCncParamReceiveTimeOut` | UInt32 | For phone profiles, the number of milliseconds to wait for a response from the phone. This time-out value is used by telephony functions that don’t need to access the network (for example, the function `TelNwkGetSelectedNetwork`).

`kCncParamResetString` | string | For modem and phone profiles, the reset string. For modem profiles, this is one of the strings in `kCncParamIntlModemResetStringList`.

`kCncParamSerialPortFlags` | UInt32 | For phone profiles, bit flags that correspond to various serial port hardware settings. See “Serial Settings Constants” on page 1560 for more information.

`kCncParamSystemFlags` | UInt32 | A bit flag representing all system flags. Currently, only bits 0 through 4 have a meaning. These correspond to the read-only bit, the invisible bit, the noneditable bit, the no details bit, and the locked bit, respectively.

`kCncParamTimeOut` | UInt32 | The amount of time in milliseconds to wait for a response when CTS is unasserted and hardware flow control is on.
### Connection Manager

#### Connection Manager Constants

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kCncParamTTCreator</td>
<td>UInt32</td>
<td>For phone profiles, the creator ID of the telephony task used by the phone driver.</td>
</tr>
<tr>
<td>kCncParamTTType</td>
<td>UInt32</td>
<td>For phone profiles, the database type for the telephony task used by the phone driver.</td>
</tr>
<tr>
<td>kCncParamVersion</td>
<td>UInt8</td>
<td>The version of the Connection Manager API under which this profile was created. The current version number is kCncProfileVersion.</td>
</tr>
<tr>
<td>kCncParamVolume</td>
<td>UInt16</td>
<td>For modem profiles, the modem volume.</td>
</tr>
</tbody>
</table>

**Compatibility** Defined only if [Connection Manager Feature Set](#) is present.

### Profile Parameter Size Constants

The size constants specify the size of each of the predefined parameters described in “Profile Parameter Constants.” The following table lists the parameter name, the size of its value, and the size constant that gives this size. These size constants are suitable for passing to `CncProfileSettingGet` and `CncProfileSettingSet`.

<table>
<thead>
<tr>
<th>Profile Parameter Name</th>
<th>Size</th>
<th>Size Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>kCncParamBaud</td>
<td>32</td>
<td>kCncParamBaudSize</td>
</tr>
<tr>
<td>kCncParamBluetoothDeviceAddr</td>
<td>8</td>
<td>kCncParamBluetoothDeviceAddrSize</td>
</tr>
<tr>
<td>kCncParamBluetoothDeviceName</td>
<td>249</td>
<td>kCncParamBluetoothDeviceNameMaxSize</td>
</tr>
<tr>
<td>kCncParamCountryIndex</td>
<td>16</td>
<td>kCncParamCountryIndexSize</td>
</tr>
</tbody>
</table>
### Connection Manager Constants

<table>
<thead>
<tr>
<th>Profile Parameter Name</th>
<th>Size</th>
<th>Size Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>kCncParamDeviceKind</td>
<td>16</td>
<td>kCncParamDeviceKindSize</td>
</tr>
<tr>
<td>kCncParamDialingMode</td>
<td>8</td>
<td>kCncParamDialingModeSize</td>
</tr>
<tr>
<td>kCncParamFlowControl</td>
<td>16</td>
<td>kCncParamFlowControlSize</td>
</tr>
<tr>
<td>kCncParamInitString</td>
<td>81</td>
<td>kCncProfileUsualInitStringMaxSize</td>
</tr>
<tr>
<td></td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>kCncParamInvisible</td>
<td>8</td>
<td>kCncParamInvisibleSize</td>
</tr>
<tr>
<td>kCncParamLocked</td>
<td>8</td>
<td>kCncParamLockedSize</td>
</tr>
<tr>
<td>kCncParamName</td>
<td>22</td>
<td>kCncParamNameMaxSize</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>kCncParamNoDetails</td>
<td>8</td>
<td>kCncParamNoDetailsSize</td>
</tr>
<tr>
<td>kCncParamNonEditable</td>
<td>8</td>
<td>kCncParamNonEditableSize</td>
</tr>
<tr>
<td>kCncParamPort</td>
<td>32</td>
<td>kCncParamPortSize</td>
</tr>
<tr>
<td>kCncParam_PSDCreator</td>
<td>32</td>
<td>kCncParam_PSDCreatorSize</td>
</tr>
<tr>
<td>kCncParam_PSDName</td>
<td>32</td>
<td>kCncParam_PSDNameSize</td>
</tr>
<tr>
<td>kCncParam_PSDType</td>
<td>32</td>
<td>kCncParam_PSDTypeSize</td>
</tr>
<tr>
<td>kCncParamReadOnly</td>
<td>8</td>
<td>kCncParamReadOnlySize</td>
</tr>
<tr>
<td>kCncParamReceiveTimeOut</td>
<td>32</td>
<td>kCncParamReceiveTimeOutSize</td>
</tr>
<tr>
<td>kCncParamResetString</td>
<td>81</td>
<td>kCncProfileClassicResetStringSize</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>kCncParamSystemFlags</td>
<td>32</td>
<td>kCncParamSystemFlagsSize</td>
</tr>
<tr>
<td>kCncParamTimeOut</td>
<td>32</td>
<td>kCncParamTimeOutSize</td>
</tr>
</tbody>
</table>
Connection Manager
Connection Manager Constants

<table>
<thead>
<tr>
<th>Profile Parameter Name</th>
<th>Size</th>
<th>Size Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>kCncParamTTCreator</td>
<td>32</td>
<td>kCncParamTTCreatorSize</td>
</tr>
<tr>
<td>kCncParamTTType</td>
<td>32</td>
<td>kCncParamTTTypeSize</td>
</tr>
<tr>
<td>kCncParamVersion</td>
<td>8</td>
<td>kCncParamVersionSize</td>
</tr>
<tr>
<td>kCncParamVolume</td>
<td>16</td>
<td>kCncParamVolumeSize</td>
</tr>
</tbody>
</table>

Compatibility Defined only if Connection Manager Feature Set is present.

Device Kind Constants
The device kind constants specify the type of connection being made. You specify the type of connection by defining a value for the kCncParamDeviceKind parameter using CncProfileSettingSet.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kCncDeviceKindSerial</td>
<td>0</td>
<td>The connection is through the serial port.</td>
</tr>
<tr>
<td>kCncDeviceKindModem</td>
<td>1</td>
<td>The connection is to a modem.</td>
</tr>
<tr>
<td>kCncDeviceKindPhone</td>
<td>2</td>
<td>The connection is to a phone.</td>
</tr>
<tr>
<td>kCncDeviceKindLocalNetwork</td>
<td>3</td>
<td>The connection is to a LAN.</td>
</tr>
</tbody>
</table>

Compatibility Defined only if Connection Manager Feature Set is present.

Profile Parameter Types
The parameter type constants specify the type of value stored for a parameter in a connection profile. If you define a new parameter using CncDefineParamID, you must use one of these constants to specify the type of data the parameter stores. The macro CncGetType can return this information for any parameter in the profile.
### Connection Manager Functions

#### CncAddProfile

**Purpose**
Adds a profile to the Connection Manager.

**Declared In**
ConnectionMgr.h

**Prototype**
Err CncAddProfile (Char *name, UInt32 port, UInt32 baud, UInt16 volume, UInt16 handShake, const Char *initString, const Char *resetString, Boolean isModem, Boolean isPulse)

**Parameters**
- <> name: Pointer to the profile name to be added. If the name is already taken in the Connection panel then a duplication number is appended to it. The name added is returned here.

### Compatibility
Defined only if [Connection Manager Feature Set](#) is present.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kCncParamSystemFlag</td>
<td>0x00</td>
<td>A system flag parameter. The Connection Manager can store up to 32 system flags. Flags are stored in a single bit and returned as a UInt8 value. The entire system flags word can be returned if you pass kCncParamSystemFlags to CncProfileSettingGet.</td>
</tr>
<tr>
<td>kCncParamUInt8</td>
<td>0x01</td>
<td>A UInt8 parameter.</td>
</tr>
<tr>
<td>kCncParamUInt16</td>
<td>0x02</td>
<td>A UInt16 parameter.</td>
</tr>
<tr>
<td>kCncParamUInt32</td>
<td>0x03</td>
<td>A UInt32 parameter.</td>
</tr>
<tr>
<td>kCncParamString</td>
<td>0x04</td>
<td>A string parameter.</td>
</tr>
<tr>
<td>kCncParamBuffer</td>
<td>0x05</td>
<td>A generic block of data.</td>
</tr>
</tbody>
</table>
Connection Manager
Connection Manager Functions

-> port  The port identification used by the profile. See “Specifying the Port” on page 100 of the Palm OS Programmer’s Companion, vol. II, Communications for more information.

-> baud  The baud rate used by the profile.

-> volume  The volume setting for the device (for Modem only).

-> handShake  Flow control setting (hardware handshaking). 0 specifies automatic (on at speeds > 2400 baud), 1 specifies always on, and 2 specifies always off.

-> initString  Pointer to the initialization string used by a modem (for Modem only).

-> resetString  Pointer to the reset string used by a modem (for Modem only).

-> isModem  true if Modem, false if Direct.

-> isPulse  true if Pulse dial, false if TouchTone.

Result  errNone  No error.

cncErrAddProfileFailed  The add operation failed.

cncErrProfileListFull  The add operation failed because the profile list is full.

cncErrConDBNotFound  The connection database is missing.

Comments  All profiles within the Connection Manager must have a unique name. The Connection Manager tries to append a duplication number to the end of the name if you specify a name that is already taken.

There is a maximum limit to the number of profiles that can be maintained by the Connection Manager. If the limit is passed, an error is returned and that profile will not be added.
Profiles that do not need certain fields may pass NULL in the place of a value.

**Compatibility**

Implemented only if [New Serial Manager Feature Set](#) is present.

If [Connection Manager Feature Set](#) is present, use [CncProfileCreate](#) instead of using this function.

CncAddProfile is still supported for backward compatibility. In Palm OS 4.0 and higher, the maximum number of profiles that can be defined has greatly increased.

**Example**

```c
AddMyProfile()
{
    Char *myConNameP;
    Err err;

    myConNameP = MemPtrNew(cncProfileNameSize);
    StrCopy(myConNameP, "Foobar");
    err = CncAddProfile(myConNameP, 'u328', 57600, 0, 0,
                        "AT&FX4", 0, true, false);

    MemPtrFree(myConNameP);
}
```

**CncDefineParamID**

**Purpose**
Macro that creates and returns a parameter ID.

**Declared In**
ConnectionMgr.h

**Prototype**

```c
CncDefineParamID (parameterRange, parameterType, parameterID)
```

**Parameters**

- parameterRange
  - Use kCncParamThirdPartiesRange to specify that this parameter is not defined by the OS.
- `parameterType`: The type of value stored for the parameter. See “Profile Parameter Types” for a list of possible values.

- `parameterID`: A unique value between 0 to 1023. The value must be unique within the profile for which you are defining the parameter.

  If you are using a `parameterType` of `kCncParamSystemFlag`, specify a value from 0 to 31 to identify which of the system flag bit is to be set.

**Result**

Returns the parameter ID as a `UInt16` value.

**Comments**

You use this macro only if you are defining your own connection profile and have a parameter that you need to define within that profile. The parameter ID immediately precedes its parameter value in the Connection Manager profile database. Because of how the database is formatted, the parameter ID must tell the Connection Manager how to interpret the next series of bytes. For this reason, the high order bits of the parameter ID include information about the type of value and whether the value is defined by the system or a third party.

**Compatibility**

Parameter IDs of this format are only used if 4.0 New Feature Set is present.

**See Also**

`CncProfileSettingSet`, `CncProfileSettingGet`
CncDeleteProfile

Purpose  Removes a profile from the Connection Manager.

Declared In  ConnectionMgr.h

Prototype  Err CncDeleteProfile (const Char *name)

Parameters  -> name  Pointer to the name of the profile to be deleted.

Result  errNone  No error.
          cncErrProfileReadOnly  The profile could not be deleted because it is read only.
          cncErrProfileNotFound  The profile could not be found.
          cncErrConDBNotFound  The connection database is missing.

Comments  The profiles that come preinstalled on the unit are read only and cannot be deleted.

Compatibility  Implemented only if New Serial Manager Feature Set is present.
                If Connection Manager Feature Set is present, use CncProfileDelete instead of using this function.
**CncGetParamType**

**Purpose**  
Macro that returns the parameter type portion of the parameter ID.

**Declared In**  
ConnectionMgr.h

**Prototype**  
CncGetParamType (parameterID)

**Parameters**  
- parameterID  
  A UInt16 that contains the parameter ID.

**Result**  
Returns a UInt16 value where bits 11 through 14 contain one of the values in “Profile Parameter Types” and the other bits are clear.

**Compatibility**  
Parameter IDs of this format are only used if 4.0 New Feature Set is present.

**See Also**  
CncProfileSettingSet, CncProfileSettingGet

---

**CncGetProfileInfo**

**Purpose**  
Returns the settings for a profile.

**Declared In**  
ConnectionMgr.h

**Prototype**  
Err CncGetProfileInfo (Char *name, UInt32 *port, UInt32 *baud, UInt16 *volume, UInt16 *handShake, Char *initString, Char *resetString, Boolean *isModem, Boolean *isPulse)

**Parameters**  
- name  
  Pointer to the name of the profile to be returned. Passing in NULL causes this function to return the settings for the profile currently selected in the Connection panel.
- port  
  Pointer to the port identifier that the profile uses.
- baud  
  Pointer to the baud rate that has been set for this profile.
Connection Manager
Connection Manager Functions

- volume Pointer to the volume of the device (applies only to modems).
- handShake Pointer to the flow control setting (hardware handshaking). 0 indicates automatic (on at speeds > 2400 baud), 1 indicates always on, and 2 indicates always off.
- initString Pointer to the initialization string for the device (applies only to modems).
- resetString Pointer to the reset string for the device (applies only to modems).
- isModem Pointer to a Boolean value: true for Modem, false for Direct.
- isPulse Pointer to a Boolean value: true for Pulse dial, false for TouchTone.

Result

- errNone No error.
- cncErrGetProfileFailed The get profile operation failed. The profile may or may not be there.
- cncErrProfileNotFound The profile could not be found
- cncErrConDBNotFound The connection database is missing.

Comments

One or more of the parameters may be set to NULL if that information is not desired.

Compatibility

Implemented only if New Serial Manager Feature Set is present.

If Connection Manager Feature Set is present, use CncProfileSettingGet with one of the “Profile Parameter Constants” instead of using this function.

Example

```
{ 
    UInt32 portID, baud;
    UInt16 openPort;
    // get port id
    err = CncGetProfileInfo("Direct Serial", &portID, &baud,
```
$0, 0, 0, 0, 0, 0);$
if(!err)
{
    // open the port
    SrmOpen(portID, baud, &openPort);
}

CncGetProfileList

**Purpose**
Returns a list of available profiles that are available through the Connection Manager.

**Declared In**
ConnectionMgr.h

**Prototype**
Err CncGetProfileList (Char ***nameListPPP, UInt16 *countP)

**Parameters**
<- nameListPPP Pointer to a pointer to a list of profile names.
<- countP Pointer to the number of profile names.
Result

errNone  No error.
cncErrGetProfileListFailed  The profile list could not be found.
cncErrConDBNotFound  The connection database is missing.

Comments  Allocation of the list is handled by the Connection Manager; deallocation is the responsibility of the calling application. Appended to the end of the list will be “-Current-”, which represents the profile currently selected in the Connection panel.

Compatibility  Implemented only if New Serial Manager Feature Set is present.

Example

//Declared globally
Char ** globalProfileList;
ListType *listP;
UInt16 globalProfileCount;

void SetConnectionList()
{
    //Get the list from the Connection Manager
    err = CncGetProfileList(&globalProfileList,
        &globalProfileCount);
    //Set the UI list
    LstSetListChoices(listP, globalProfileList,
        globalProfileCount);
}

void StopApplication()
{
    UInt16 i;

    //Deallocate the connection list
    For(i = 0; i < globalProfileCount; i++)
        MemPtrFree(globalProfileList[ i ]);
    MemPtrFree(globalProfileList);
}
CncGetSystemFlagBitnum

**Purpose**  
Macro that returns the number uniquely identifying a system flag parameter.

**Declared In**  
ConnectionMgr.h

**Prototype**  
CncGetSystemFlagBitnum (parameterID)

**Parameters**  
-> parameterID  
The UInt16 containing the parameter ID.

**Result**  
Returns the ID of the system flag parameter, which is a value from 0 to 31.

**Compatibility**  
Parameter IDs of this format are only used if Connection Manager Feature Set is present.

**See Also**  
CncProfileSettingSet, CncProfileSettingGet

CncGetTrueParamID

**Purpose**  
Macro that returns the portion of the parameter ID that uniquely identifies the parameter.

**Declared In**  
ConnectionMgr.h

**Prototype**  
CncGetTrueParamID (parameterID)

**Parameters**  
-> parameterID  
A UInt16 containing the parameter ID.

**Result**  
Returns a UInt16 containing just the parameter ID. The high-order bits, which specify the parameter type and address space, are clear.

**Compatibility**  
Parameter IDs of this format are only used if Connection Manager Feature Set is present.

**See Also**  
CncProfileSettingSet, CncProfileSettingGet
CncIsFixedLengthParamType

Purpose
Macro that specifies whether the parameter value is fixed length or variable length.

Declared In
ConnectionMgr.h

Prototype
CncIsFixedLengthParamType (parameterID)

Parameters
- parameterID
  A UInt16 containing the parameter ID.

Result
Returns true if the parameter is a fixed length parameter type such as UInt32, or false if it is a variable length type.

Compatibility
Parameter IDs of this format are only used if Connection Manager Feature Set is present.

See Also
CncProfileSettingSet, CncProfileSettingGet

CncIsSystemFlags

Purpose
Macro that returns whether the parameter value is a system flag.

Declared In
ConnectionMgr.h

Prototype
CncIsSystemFlags (parameterID)

Parameters
- parameterID
  The UInt16 containing the parameter ID.

Result
Returns true if the parameter type is a system flag. Returns false otherwise.

Compatibility
Parameter IDs of this format are only used if Connection Manager Feature Set is present.

See Also
CncProfileSettingSet, CncProfileSettingGet
CncIsSystemRange

**Purpose**  
Macro that specifies whether the parameter is in the system range or in the third-party range.

**Declared In**  
ConnectionMgr.h

**Prototype**  
CncIsSystemRange (parameterID)

**Parameters**  
- `parameterID`  
  A UInt16 containing the parameter ID.

**Result**  
Returns `true` if the parameter ID is defined by Palm OS, or `false` if it is defined by a third party.

**Compatibility**  
Parameter IDs of this format are only used if Connection Manager Feature Set is present.

**See Also**  
CncProfileSettingSet, CncProfileSettingGet

CncIsThirdPartiesRange

**Purpose**  
Macro that specifies whether the parameter is defined by a third party.

**Declared In**  
ConnectionMgr.h

**Prototype**  
CncIsThirdPartiesRange (parameterID)

**Parameters**  
- `parameterID`  
  A UInt16 containing the parameter ID.

**Result**  
Returns `true` if the parameter is a third-party parameter, or `false` if it is a system parameter.

**Compatibility**  
Parameter IDs of this format are only used if Connection Manager Feature Set is present.

**See Also**  
CncProfileSettingSet, CncProfileSettingGet
**CncIsVariableLengthParamType**

**Purpose**
Macro that returns whether the parameter value is a variable-length type.

**Declared In**
ConnectionMgr.h

**Prototype**
CncIsVariableLengthParamType (parameterID)

**Parameters**
- parameterID  
  A UInt16 containing the parameter ID.

**Result**
Returns true if the parameter is a variable-length string or a buffer or false if it holds a fixed-length type such as an integer.

**Compatibility**
Parameter IDs of this format are only used if Connection Manager Feature Set is present.

**See Also**
CncProfileSettingSet, CncProfileSettingGet

---

**CncProfileCloseDB**

**Purpose**
Closes the Connection Manager profile database.

**Declared In**
ConnectionMgr.h

**Prototype**
Err CncProfileCloseDB (void)

**Parameters**
None.

**Result**
- errNone  
  No error.
- kCncErrDBAccessFailed
  The database could not be closed or this is a reference counting error.

**Comments**
Use CncProfileOpenDB and CncProfileCloseDB as an optimization if you make several Connection Manager calls in succession. All Connection Manager calls open the profile database
when they begin and close the database when they are finished. The Connection Manager maintains a reference count that tells it whether the database is open. If you call CncProfileOpenDB before making another Connection Manager call, the next call does not open or close the database. This saves your application the overhead of opening and closing the database each time a call is made.

**Compatibility**  
Implemented only if [Connection Manager Feature Set](#) is present.

### CncProfileCount

**Purpose**  
Returns the number of connection profiles currently defined in the Connection Manager profile database.

**Declared In**  
ConnectionMgr.h

**Prototype**  
Err CncProfileCount (UInt16 *profilesCountP)

**Parameters**

- `<profilesCountP`  
The number of profiles.

**Result**

- `errNone`  
  No error.
- `kCncErrDBAccessFailed`  
  The profile database could not be opened.

**Compatibility**  
Implemented only if [Connection Manager Feature Set](#) is present.

**See Also**  
[CncGetProfileList](#)
CncProfileCreate

**Purpose**
Adds a profile record to the Connection Manager profile database.

**Declared In**
ConnectionMgr.h

**Prototype**
Err CncProfileCreate (CncProfileID *profileIdP)

**Parameters**
profileIdP
Upon return, the unique ID of the new profile.

**Result**
errNone
No error.
kCncErrDBAccessFailed
The profile database could not be opened.
a Data Manager error
The new record could not be created.

**Comments**
This function creates a new empty record in the Connection Manager profile database. To populate the profile, use CncProfileSettingSet to set parameter values, including the profile name. Use CncDefineParamID if you need to store information unique to your profile.

**Compatibility**
Implemented only if Connection Manager Feature Set is present.

**See Also**
CncAddProfile, CncProfileDelete

CncProfileDelete

**Purpose**
Deletes a profile.

**Declared In**
ConnectionMgr.h

**Prototype**
Err CncProfileDelete (CncProfileID profileId)

**Parameters**
profileId
The ID of the profile to delete.

**Result**
errNone
No error.
kCncErrDBAccessFailed
The profile database could not be opened, or
the record could not be deleted.

kCncErrProfileParamNotFound
The database does not contain a profile with the
specified ID.

Comments
The profiles that come preinstalled on the unit are read only and
cannot be deleted.

Compatibility
Implemented only if Connection Manager Feature Set is present.

See Also
CncDeleteProfile, CncProfileCreate

CncProfileGetCurrent

Purpose
Returns the ID of the currently selected profile in the Connection
panel.

Declared In
ConnectionMgr.h

Prototype
Err CncProfileGetCurrent
(CncProfileID *profileIdP)

Parameters
<-profileIdP The ID of the current profile.

Result
errNone No error.

kCncErrDBAccessFailed
The profile database could not be opened.

Compatibility
Implemented only if Connection Manager Feature Set is present.

See Also
CncProfileGetIDFromIndex, CncProfileGetIDFromName,
CncProfileGetIndex, CncProfileSetCurrent
**CncProfileGetIDFromIndex**

**Purpose**
Returns the profile ID given its index into the Connection Manager profile database.

**Declared In**
ConnectionMgr.h

**Prototype**
```c
Err CncProfileGetIDFromIndex (UInt16 index, CncProfileID *profileIdP)
```

**Parameters**
- `index` The index of the Connection Manager profile.
- `profileIdP` The ID of the Connection Manager profile.

**Result**
- `errNone` No error.
- `kCncErrDBAccessFailed` The profile database could not be opened.
- `kCncErrProfileParamNotFound` No profile at that index.

**Compatibility**
Implemented only if [Connection Manager Feature Set](#) is present.

**See Also**
[CncProfileGetIDFromName](#), [CncProfileGetCurrent](#), [CncProfileGetIndex](#)
CncProfileGetIDFromName

Purpose
Returns the profile ID given its name.

Declared In
ConnectionMgr.h

Prototype
Err CncProfileGetIDFromName
(const Char *profileNameP,
CncProfileID *profileIdP)

Parameters
-> profileNameP  The name of the profile. The name is displayed
  in a pop-up list in the Connection panel. If you
  pass the string "-Current-", this function
  returns the ID of the current profile.

<- profileIdP  The profile ID.

Result
errNone  No error.

kCncErrDBAccessFailed  The profile database could not be opened.

kCncErrProfileParamNotFound  No profile with the specified name.

Compatibility
Implemented only if Connection Manager Feature Set is present.

See Also
CncProfileGetCurrent, CncProfileGetIDFromIndex,
CncProfileGetIndex

CncProfileGetIndex

Purpose
Returns the index of the profile given its ID.

Declared In
ConnectionMgr.h

Prototype
Err CncProfileGetIndex (CncProfileID profileId,
UInt16 *indexP)

Parameters
-> profileId  The profile ID.
The index of the profile’s record in the Connection Manager profile database.

Result
- errNone: No error.
- kCncErrDBAccessFailed: The profile database could not be opened.
- kCncErrProfileParamNotFound: No profile with the specified ID.

Compatibility
Implemented only if Connection Manager Feature Set is present.

See Also
CncProfileGetIDFromIndex

**CncProfileOpenDB**

Purpose
Opens the Connection Manager profile database.

Declared In
ConnectionMgr.h

Prototype
Err CncProfileOpenDB (void)

Parameters
None

Result
- errNone: No error.
- kCncErrDBAccessFailed: The profile database could not be opened.

Comments
Use CncProfileOpenDB and CncProfileCloseDB as an optimization if you make several Connection Manager calls in succession. All Connection Manager calls open the profile database when they begin and close the database when they are finished. The Connection Manager maintains a reference count that tells it whether the database is open. If you call CncProfileOpenDB before making another Connection Manager call, the next call does not open or close the database. This saves your application the overhead of opening and closing the database each time a call is made.
The Connection Manager profile database is created if it does not exist.

**Compatibility**  
Implemented only if [Connection Manager Feature Set](#) is present.

---

### CncProfileSetCurrent

**Purpose**  
Sets the current profile.

**Declared In**  
ConnectionMgr.h

**Prototype**  
Err CncProfileSetCurrent (CncProfileID profileId)

**Parameters**  
- profileId  
The ID of the profile to be made current.

**Result**  
- errNone  
  No error.
- kCncErrDBAccessFailed  
The profile database could not be opened.

**Comments**  
The current profile is the profile that is used for the next network connection attempt.

**Compatibility**  
Implemented only if [Connection Manager Feature Set](#) is present.

**See Also**  
CncProfileGetCurrent
CncProfileSettingGet

**Purpose**
Obtains a value stored in one of the Connection Manager profiles.

**Declared In**
ConnectionMgr.h

**Prototype**
Err CncProfileSettingGet (CncProfileID profileId, Uint16 paramId, void *paramBufferP, Uint16 *ioParamSizeP)

**Parameters**
- `profileId` The ID of the profile from which to obtain a parameter value.
- `paramId` The ID of the parameter to obtain. See “Profile Parameter Constants” for a list of the parameters used in the profiles that come preinstalled on the device.
- `paramBufferP` A pointer to a buffer into which to write the parameter value. If the parameter stores a variable-sized value, you can determine the necessary size by passing NULL for paramBufferP. Upon return, paramSize contains the required size.
- `ioParamSizeP` On input, a pointer to the size of the buffer into which to write the parameter. On output, points to the number of bytes written to paramBufferP.

**Result**
- `errNone` No error.
- `kCncErrDBAccessFailed` The profile database could not be opened.
- `kCncErrProfileGetParamFailed` The Connection Manager failed to obtain the value of the parameter.
- `kCncErrProfileBadSystemFlagBitnum` An attempt was made to obtain the value of a system flag that is undefined.
kCncErrProfileBadParamSize
The paramBufferP buffer is too small.
ioParamSizeP contains the correct size for the buffer.

kCncErrProfileParamNotFound
The specified parameter is not defined in the profile.

Compatibility
Implemented only if Connection Manager Feature Set is present.

See Also CncProfileSettingSet, CncGetProfileInfo

CncProfileSettingSet

Purpose
Sets a parameter value in the specified profile.

Declared In ConnectionMgr.h

Prototype
Err CncProfileSettingSet (CncProfileID iProfileId, Uint16 paramId, const void *paramBufferP, Uint16 paramSize)

Parameters
- iProfileId The ID of the profile.
- paramId The ID of the parameter. See “Profile Parameter Constants” for a list of the parameters defined in the preinstalled connection profiles.
- paramBufferP A pointer to the value to set for this parameter.
- paramSize The size of the buffer passed in paramBufferP. See “Profile Parameter Size Constants.”

Result
errNone No error.
kCncErrDBAccessFailed The profile database could not be opened.
Connection Manager

Connection Manager Functions

kCncErrProfileParamNotFound
No profile with the specified ID.

kCncErrProfileSetParamFailed
The Connection Manager failed to set the value of the parameter.

kCncErrProfileBadParamSize
The paramBufferP buffer is too small. Most likely, the passed in size does not allow space for a string parameter’s null terminator.

kCncErrProfileParamNameHasChange
An attempt was made to set the profile name to a name that is already used. The Connection Manager appends a duplication number to the end of the name and returns this error. You should use CncProfileSettingGet to find out the new name.

Compatibility
Implemented only if Connection Manager Feature Set is present.

See Also
CncDefineParamID, CncProfileSettingGet
This chapter describes the Exchange Manager API declared in the header file ExgMgr.h and the Exchange Local Library API declared in the header file ExgLocalLib.h. It discusses the following topics:

- Exchange Manager Data Structures
- Exchange Manager Constants
- Exchange Manager Functions
- Application-Defined Functions

For more information on the Exchange Manager, see the chapter “Object Exchange” on page 1 of Palm OS Programmer’s Companion, vol. II, Communications.

Exchange Manager Data Structures

**ExgAskResultType**

The ExgAskResultType enum defines possible values for the result field of the sysAppLaunchCmdExgAskUser launch code parameter block.

```c
typedef enum {
    exgAskDialog,
    exgAskOk,
    exgAskCancel
} ExgAskResultType;
```
Value Descriptions

**exgAskDialog** The Exchange Manager should display a dialog that prompts the user to confirm the receipt of data. See ExgDoDialog.

**exgAskOk** Accept the data.

**exgAskCancel** Reject the data.

---

**ExgGoToType**

The ExgGoToType structure defines the goToParams field of the ExgSocketType structure. Applications that want to be launched after the data is received place their creator IDs in the goToCreator field and define the goToParams field. The values in this structure are copied to the sysAppLaunchCmdGoto launch code’s parameter block.

```c
typedef struct {
    UInt16 dbCardNo;
    LocalID dbID;
    UInt16 recordNum;
    UInt32 uniqueID;
    UInt32 matchCustom;
} ExgGoToType;
```

Field Descriptions

**dbCardNo** The card number of the database that contains the added record.

**dbID** The local ID of the database that contains the added record.

**recordNum** The index of the record that was added.

**uniqueID** The unique ID of the record that was added. This field is not used.

**matchCustom** Application-specific information.
ExgLocalSocketInfoType

The ExgLocalSocketInfoType structure identifies information specific to the Local Exchange Library. The socketRef field of the ExgSocketType structure is set to this structure when you send and receive data using the Local Exchange Library. The Local Exchange Library creates this structure if it does not already exist. You only need to create it if you want to supply non-default values for the noAsk or previewInfoP fields.

```c
typedef struct {
    Boolean freeOnDisconnect;
    Boolean noAsk;
    ExgPreviewInfoType *previewInfoP;
    FileHand tempFileH;
    Err err;
    ExgLocalOpType op;
} ExgLocalSocketInfoType;
```

**Field Descriptions**

**freeOnDisconnect**  Whether the structure is freed when the ExgDisconnect call is made. The default is true. In general, code that allocates a structure should be responsible for freeing that structure. Therefore, if you have allocated ExgLocalSocketInfoType, you should set this field to false and explicitly freee the structure when you are finished with it.

**noAsk**  Set to true to disable the display of the exchange dialog. If you want to, for example, create a vCalendar object and send it to the Datebook application in response to a user command, you probably want to set noAsk to true so that the user does not have to confirm the receipt of the data they just requested you to send.
Exchange Manager
Exchange Manager Data Structures

previewInfoP A pointer to an ExgPreviewInfoType structure, used to display a preview of the data. If you wanted to simply use another application to help display data, you would create and initialize this structure.

tempFileH A temporary buffer that the Local Exchange Library uses. Do not set this field directly; the Local Exchange Library should set it.

err The error code returned from the Local Exchange Library. Do not set this field directly; the Local Exchange Library should set it.

op The operation in progress. Do not set this field directly. The Local Exchange Library sets this field to one of the following:

- exgLocalOpNone No operation in progress.
- exgLocalOpPut A send is in progress.
- exgLocalOpAccept A receive is in progress.
- exgLocalOpGet A get is in progress.
- exgLocalOpGetSender The library is receiving information from the sending application during a get operation.
The **ExgPreviewInfoType** structure provides information to the [ExgNotifyPreview](#) function. The ExgNotifyPreview function uses this information to have the application display a preview of the data to be received in the exchange dialog.

```c
typedef struct {
    UInt16         version;
    ExgSocketType  *socketP;
    UInt16         op;
    Char           *string;
    UInt32         size;
    RectangleType  bounds;
    UInt16         types;
    Err            error;
} ExgPreviewInfoType;
```

### Field Descriptions

**-> version**  Set this field to 0 to specify version 0 of this structure.

**-> socketP**  A pointer to the socket structure (see [ExgSocketType](#)). The libraryRef field must identify the exchange library from which preview data should be received, and the target, type, or name field should be defined as well.

**-> op**  One of the following constants:

- `exgPreviewDialog`  Display a modal dialog containing the preview. This constant is only used in situations where one application launches another to display data.

- `exgPreviewDraw`  The preview is a graphic.

- `exgPreviewLongString`  The preview is a long string.
Applications can define and use their own constants for the preview operation. Operations specific to an application are numbered starting at `exgPreviewFirstUser` and should be no greater than `exgPreviewLastUser`.

**Compatibility**  
This structure is only defined if [4.0 New Feature Set](#) is present.
**ExgSocketType**

The `ExgSocketType` structure defines an Exchange Manager socket, which is passed to most Exchange Manager functions. The `ExgSocketPtr` type points to a `ExgSocketType` structure.

```c
typedef struct ExgSocketType {
    Uint16 libraryRef;
    Uint32 socketRef;
    Uint32 target;
    Uint32 count;
    Uint32 length;
    Uint32 time;
    Uint32 appData;
    Uint32 goToCreator;
    ExgGoToType goToParams;
    Uint16 localMode:1;
    Uint16 packetMode:1;
    Uint16 noGoTo:1;
    Uint16 noStatus:1;
    Uint16 preview:1;
    Uint16 reserved:11;
    Char *description;
    Char *type;
    Char *name;
} ExgSocketType;
```

Note that when data is received, some of the fields in this structure may not have values. When you are sending data, it is recommended that you provide values for all of these fields, but you should not rely on receiving values for the fields marked optional.
Field Descriptions

**libraryRef**  
The exchange library in use. When an application or library receives a socket, this field is already assigned.

When sending data, applications may identify the exchange library they want to connect with by providing a URL in the name field. If so, they should use 0 for the libraryRef field. The Exchange Manager then determines which library corresponds to the URL and assigns the libraryRef field. See the Comments in the ExgPut function description for more information.

**socketRef**  
The connection identifier. This value is supplied by the exchange library when a connection is established. It contains any necessary library-specific data.

**target**  
The creator ID of the application that should receive the message.

**count**  
The number of objects in this connection, usually 1 (optional).

**length**  
The total byte count for all objects being sent (optional).

**time**  
The last modified time of object (optional).

**appData**  
Application-specific information (optional).

**goToCreator**  
The creator ID of the application to launch using the sysAppLaunchCmdGoto launch code after the item is received if noGoTo is 0. The value is assigned by the application that receives the object. See the Comments section in ExgDisconnect for more information.

**goToParams**  
If goToCreator is specified, then this field contains data that is copied into the launch code’s parameter block. See ExgGoToType.
localMode Set to 1 to exchange with local device only. A localMode of 1 is equivalent to specifying a URL with the exgLocalPrefix. Set to 0 to enable an exchange with a remote machine. The default is 0.

packetMode Set to 1 to use connectionless packet mode (Ultra). The default is 0. Ultra mode is not currently supported.

noGoTo Set to 1 to disable launching the application with sysAppLaunchCmdGoto. The default is 0.

noStatus If true, the exchange library should not display a progress dialog. If false, the library can display a progress dialog. The default is false.

The Exchange Manager sets and clears this bit at various times while data is received. Applications may also want to set this bit if they use the Local Exchange Library and want to prevent the progress dialog from being displayed during a send.

preview If true, a preview is in progress. The ExgNotifyPreview function sets this bit while the preview takes place and clears it when the preview is finished. Exchange libraries should not discard any data while a preview is in progress because the full data must be sent later if the receiving user accepts it.

reserved Reserved system flags.

description A pointer to the text description of the object (optional).

type A pointer to the MIME type of the object (optional).
Exchange Manager
Exchange Manager Constants

name

The name of the object being sent. This can be a URL whose scheme identifies the exchange library to connect with.

If the name has a colon, it is treated as a URL.

Compatibility

The noGoTo and noStatus flags are only defined if 3.5 New Feature Set is present, and the noStatus flag has no effect unless 4.0 New Feature Set is present. The preview flag is only defined if 4.0 New Feature Set is present.

Exchange Manager Constants

Registry ID Constants

The registry ID constants are used in the Exchange Manager registry. Exchange libraries register for the URL prefixes they handle, and applications register for the types of data they receive. The registry ID constants specify which type of data is being registered for.
Compatibility

The exgRegCreatorID and exgRegSchemeID constants are only defined if 4.0 New Feature Set is present.

Predefined URL Schemes

The Exchange Manager provides these predefined URL schemes, for which exchange libraries can register.
### Exchange Manager

#### Exchange Manager Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exgBeamScheme</td>
<td>&quot;_beam&quot;</td>
<td>The URL scheme for Beam commands. By default, the IR Library handles this scheme.</td>
</tr>
<tr>
<td>exgSendScheme</td>
<td>&quot;_send&quot;</td>
<td>The URL scheme for Send commands. The purpose of the Send command is to provide a choice of transport mechanisms to the user; therefore, any exchange library that sends data should register for this scheme.</td>
</tr>
<tr>
<td>exgLocalScheme</td>
<td>&quot;_local&quot;</td>
<td>The URL scheme for the Local Exchange Library.</td>
</tr>
</tbody>
</table>

#### Compatibility

These constants are only defined if 4.0 New Feature Set is present.

### Predefined URL Prefixes

The Exchange Manager provides the following prefixes, which can be used to construct URLs appropriate for the name field of the `ExgSocketType` structure. When sending data, applications provide a URL to identify the exchange library that should transport the data.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exgBeamPrefix</td>
<td>(exgBeamScheme &quot;:&quot;)</td>
<td>The URL to beam data.</td>
</tr>
<tr>
<td>exgSendPrefix</td>
<td>(&quot;?&quot; exgSendScheme &quot;:&quot;)</td>
<td>A URL for the general Send command. Because this URL begins with a question mark (?), the Exchange Manager displays a dialog with a list of exchange libraries registered for the <code>exgSendScheme</code>. The user then chooses the desired exchange library.</td>
</tr>
</tbody>
</table>
Compatibility
These constants are only defined if [4.0 New Feature Set](#) is present.

## Exchange Manager Functions

### ExgAccept

**Purpose**
Accepts a connection from a remote device.

**Declared In**
ExgMgr.h

**Prototype**
Err ExgAccept (ExgSocketType *socketP)

**Parameters**
- `socketP` A pointer to the socket structure (see ExgSocketType).

**Result**
Returns one of the following error codes:
- `errNone` Success
- `exgErrBadLibrary` Couldn’t find default exchange library
- `exgErrNotSupported` A preview is in progress, and the exchange library identified by `libraryRef` doesn’t support preview mode

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exgSendBeamPrefix</td>
<td>&quot;?&quot; exgSendScheme &quot;;&quot; exgBeamScheme &quot;:&quot;</td>
<td>A URL for the general Send command. The Exchange Manager displays a dialog with a list of exchange libraries registered for either the exgSendScheme or the exgBeamScheme.</td>
</tr>
<tr>
<td>exgLocalPrefix</td>
<td>(exgLocalScheme &quot;:&quot;)</td>
<td>The URL for using the Local Exchange Library.</td>
</tr>
</tbody>
</table>
Other error codes depend on the exchange library. Displays a fatal error message if socketP does not have a libraryRef specified.

Comments

Applications call this function when launched with the sysAppLaunchCmdExgReceiveData or sysAppLaunchCmdExgPreview launch code. The launch code contains socketP in its parameter block. Applications should pass this socket to ExgAccept to accept the connection, then call ExgReceive one or more times to receive the data, and then call ExgDisconnect to disconnect.

NOTE: Don’t create the socket on the receiving side of an exchange. The socket is passed to you in the command parameter block of the sysAppLaunchCmdExgReceiveData or sysAppLaunchCmdExgPreview launch code.

Compatibility

Implemented only if 3.0 New Feature Set is present. Preview mode is supported only if 4.0 New Feature Set is present.

See Also

ExgConnect, ExgPut, ExgGet

ExgConnect

Purpose

Establishes a connection with a remote socket.

Declared In

ExgMgr.h

Prototype

Err ExgConnect (ExgSocketType *socketP)

Parameters

-> socketP  A pointer to the socket structure (see ExgSocketType). Specify either a value for the libraryRef field or a URL in the name field. libraryRef should be 0 if the name field contains a URL.

Result

Returns one of the following error codes:
errNone  Success
exgErrBadLibrary  Couldn’t find exchange library
exgErrNotSupported  The library doesn’t support the operation specified in socketP
exgErrUserCancel  The user cancelled the connection operation
exgMemError  There isn’t enough free memory to respond to the request
exgErrNotEnoughPower  The battery does not have enough power to perform the operation

Other error codes depend on the exchange library.

Comments
Applications can call this function to initiate a connection for sending multiple objects or for performing two-way communications. Some exchange libraries support sending multiple objects but do not support this call. See “Sending Multiple Objects” on page 17 of Palm OS Programmer’s Companion, vol. II, Communications for more information.

Before calling this function, the application must initialize the socketP parameter. The socket should identify the exchange library to connect with by providing either a library reference number in the libraryRef field or a URL in the name field. The default exchange library registered for that type of URL handles the connection.

To provide users with a choice of transport mechanisms, specify a URL that begins with a question mark (?). The Exchange Manager displays a dialog with a list of all exchange libraries that respond to URLs of the specified type. If only one exchange library is registered for this URL scheme, no dialog is displayed.

For example, many applications on Palm OS® 4.0 or higher support a Send command. This command generates a URL with the prefix exgSendPrefix (see Predefined URL Prefixes). The Exchange Manager displays a dialog containing a list of libraries registered for that URL scheme. The user selects an exchange library, and that library’s ExgLibConnect function is called.
If the library is not specified by either URL or library reference number (in the libraryRef field), the Exchange Manager by default uses the IR Library; however, if the localMode flag is set, the Local Exchange Library is used instead.

In addition to specifying the library, you can set the count field in socketP before making this call to indicate the number of objects that are going to be sent. Use a count of 0 if the number of objects isn’t known in advance.

If no error is returned from ExgConnect, applications can follow this call either by sending multiple objects or requesting data from the remote device or both. To send an object, call ExgPut at the beginning of each object and call ExgSend one or more times per object to send the data. To request data from the remote device, use ExgGet (and then use ExgReceive to receive the requested data). You can use these calls in combination with each other to support two-way communications. After all of the objects have been sent and received, call ExgDisconnect to disconnect.

**IMPORTANT:** Not all exchange libraries support the sending of multiple objects or using ExgGet to request data.

**Compatibility**

Implemented only if 3.0 New Feature Set is present. ExgConnect was for system use only until the release of Palm OS 4.0. Multiple object sending and identifying exchange libraries by URL are supported only if 4.0 New Feature Set is present. On earlier releases, this function is an alias for the ExgPut function.

**See Also** ExgPut, ExgAccept, ExgGet
**ExgControl**

**Purpose** Requests that an exchange library perform an operation.

**Declared In** ExgMgr.h

**Prototype**
```
Err ExgControl (ExgSocketType *socketP,
                Uint16 op, void *valueP, Uint16 *valueLenP)
```

**Parameters**
- `socketP` A pointer to the socket structure (see ExgSocketType). Specify either a value for the libraryRef field or a URL in the name field. libraryRef should be 0 if the name field contains a URL.
- `op` A constant identifying the operation that the exchange library should perform. See the Comments section for more information.
- `valueP` Upon entry, a parameter that the exchange library requires to perform the operation, if any. Most operations do not require an input parameter. Upon return, contains the result of the operation.
- `valueLenP` The size of the valueP buffer. The size is updated upon return to show the actual length of the content returned.

**Result** Returns one of the following error codes:
- **errNone** Success
- **exgErrBadLibrary** Couldn’t find the requested exchange library
- **exgErrNotSupported** The exchange library does not support the requested operation

Other error codes depend on the exchange library.
The Exchange Manager uses this function to request information from the exchange library. Applications may also call this function.

The Exchange Manager defines and uses a set of operation constants that it might send to any exchange library. These constants begin with the prefix `exgLibCtlGet`. The type of the variable pointed to by `valueP` depends on the type of operation to be performed. Table 57.1 lists and describes the predefined Exchange Manager operations.

**Table 57.1 ExgControl operations for all exchange libraries**

<table>
<thead>
<tr>
<th>Operation <code>exgLibCtlGet...</code> value</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preview</td>
<td>Boolean. Output only.</td>
<td>Returns <code>true</code> if the exchange library supports preview mode or <code>false</code> if not. If the exchange library does not respond to this operation, it is assumed to support preview mode.</td>
</tr>
<tr>
<td>Title</td>
<td>String buffer of size <code>exgTitleBufferSize</code> bytes. Output only.</td>
<td>Returns the name of the exchange library as it should appear in the Send dialog. All exchange libraries must respond to this operation.</td>
</tr>
<tr>
<td>Version</td>
<td>UInt16. Output only.</td>
<td>Returns the version of the exchange library API that this library implements. The constant <code>exgLibAPIVersion</code> defines the current version number. If the exchange library does not respond to this operation, the library supports the version of the Exchange Library API defined in Palm OS 4.0.</td>
</tr>
</tbody>
</table>

An exchange library may also define its own operations. For example, the IR Library supports operations to enable or disable beaming, to set the baud rates, or to use the serial port (see “IR Control Constants” on page 1378). The SMS Library supports operations that allow you to set the SMS preferences for sending messages or to manipulate multipart messages (see “SMS Control...” on page 1381).
Constants” on page 2232). Operations specific to an exchange library are numbered starting at exgLibCtlSpecificOp.

The socketP passed to this function must identify an exchange library either using the libraryRef field or using a URL in the name field. The Comments section in ExgConnect describes how an application should identify the exchange library.

Compatibility Implemented only if 4.0 New Feature Set is present.

**ExgDBRead**

**Purpose** Converts a Palm OS database from its internal format and writes it to storage RAM.

**Declared In** ExgMgr.h

**Prototype**

```c
Err ExgDBRead (ExgDBReadProcPtr readProcP, ExgDBDeleteProcPtr deleteProcP, void* userDataP, LocalID* dbIDP, UInt16 cardNo, Boolean* needResetP, Boolean keepDates)
```

**Parameters**

- `readProcP` A pointer to a function that reads in the database and passes it to ExgDBRead. See ExgDBReadProcPtr for details.

- `deleteProcP` A pointer to a function that is called if a database with an identical name already exists on the device. See ExgDBDeleteProcPtr for details.

- `userDataP` A pointer to any data you want to pass to either the readProcP or deleteProcP functions. Often, this parameter is used to pass the ExgSocketType that is required by many Exchange Manager functions.

- `dbIDP` The ID of the database that ExgDBRead created on the local device.

- `cardNo` The number of the card on which the database was stored by ExgDBRead.
Exchange Manager
Exchange Manager Functions

<- needResetP Set to true by ExgDBRead if the
dmHdrAttrResetAfterInstall attribute
bit is set in the received database.

-> keepDates Specify true to retain the creation,
modification, and last backup dates as set in the
received database header. Specify false to
reset these dates to the current date.

Result Returns errNone if successful; otherwise, returns one of the data
manager error codes (dmErr...) or a callback-specific error code.
(If the readProcP function returns an error, it is also returned by
ExgDBRead.)

Comments This function converts data received from an exchange library or
from any other transport mechanism into a Palm OS database and
stores that database in the storage heap. It is not required that you
use this function in conjunction with Exchange Manager calls. That
is, it’s possible to use this function to perform other operations, such
as converting a database created on the desktop computer to a Palm
OS formatted database in the storage heap.
The primary use of this function, however, is to receive a database that has been beamed onto the device. In this case, call \texttt{ExgDBRead} in response to the launch code \texttt{sysAppLaunchCmdExgReceiveData} after calling \texttt{ExqAccept} to accept the connection. Place the call to \texttt{ExqReceive} in the read callback function you passed as the \texttt{readProcP} parameter. Pass the \texttt{ExqSocketType} structure returned from \texttt{ExgAccept} in the \texttt{userDataP} parameter so that you have access to it in the read callback function.

The read callback function performs the actual reading of data. \texttt{ExgDBRead} calls the read callback function multiple times. Each time, the \texttt{sizeP} parameter contains the number of bytes \texttt{ExgDBRead} expects the data returned in \texttt{dataP} to contain. It’s important for the read callback function to set the number of bytes (in \texttt{sizeP}) that it actually placed in \texttt{dataP} if it’s not the same as what \texttt{ExgDBRead} expected. \texttt{ExgDBRead} stops calling the read callback function after 0 is returned in \texttt{sizeP}.

The callback function you pass in \texttt{deleteProcP} handles the case where the database being read already exists on the device. It is called only in that circumstance. The callback function may want to close the database if it is open, change the existing database’s name, or delete the existing database to allow an overwrite. See \texttt{ExgDBDeleteProcPtr} for more information.

\textbf{Compatibility} \hspace{1cm} Implemented only if \texttt{3.0 New Feature Set} is present.

\textbf{See Also} \hspace{1cm} \texttt{ExgDBWrite}
**ExgDBWrite**

**Purpose**
Converts a given Palm OS database from its internal format on the local device and writes it using a function you supply.

**Declared In**
ExgMgr.h

**Prototype**
```c
Err ExgDBWrite (ExgDBWriteProcPtr writeProcP, void* userDataP, const char* nameP, LocalID dbID, Uint16 cardNo)
```

**Parameters**
- `-> writeProcP` A pointer to a function that writes out the database identified by `dbID`. See `ExgDBWriteProcPtr` for details.
- `-> userDataP` A pointer to any data you want to pass to the `writeProcP` function. Often, this parameter is used to pass the `ExgSocketType` that is required by many Exchange Manager functions.
- `-> nameP` A pointer to the name of the database that you want `ExgDBWrite` to write. This database is passed to `writeProcP`.
- `-> dbID` The ID of the database that you want `ExgDBWrite` to pass to `writeProcP`. If you don’t supply an ID, then `nameP` is used to search for the database by name.
- `-> cardNo` The number of the card on which to look for the database identified by `nameP`.

**Result**
Returns `errNone` if successful; otherwise, returns one of the data manager error codes (`dmErr...`) or a callback-specific error code. (If the `writeProcP` function returns an error, it is also returned by `ExgDBWrite`.)

**Comments**
This function converts a Palm OS formatted database on the storage heap into a stream of bytes that can be sent over the Internet or over
any other transport mechanism. It is not required that you use this function in conjunction with Exchange Manager calls.

The primary use of this function, however, is to write a database that is going to be beamed onto another device. In this case, call ExgDBWrite after establishing the connection with ExgPut. Place the call to ExgSend in the write callback function you passed as the writeProcP parameter. Pass the ExgSocketType structure returned from ExgSend in the userDataP parameter so that you have access to it in the write callback function.

The write callback function performs the actual writing of data. ExgDBWrite calls the write callback function multiple times. Each time, the sizeP parameter contains the number of bytes of dataP that are to be written. If the write callback function didn’t handle it all, it’s important that it set in sizeP the number of bytes that it did handle successfully. ExgDBWrite stops calling the write callback function after 0 is returned in sizeP.

**Compatibility**
Implemented only if 3.0 New Feature Set is present.

**See Also**
ExgDBRead

### ExgDisconnect

**Purpose**
Terminates an Exchange Manager transfer and disconnects.

**Declared In**
ExgMgr.h

**Prototype**
```c
Err ExgDisconnect (ExgSocketType *socketP,
    Err error)
```

**Parameters**
- socketP
  A pointer to the socket structure (see ExgSocketType) identifying the connection to terminate.
-> error Any error that occurred. This parameter tells the exchange library why the connection is being broken. Normally the error code from ExgSend or ExgReceive is passed in here.

**Result**

Returns one of the following error codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errNone</td>
<td>Success</td>
</tr>
<tr>
<td>exgErrBadLibrary</td>
<td>Couldn’t find default exchange library</td>
</tr>
<tr>
<td>exgMemError</td>
<td>Couldn’t read data to send</td>
</tr>
<tr>
<td>exgErrUserCancel</td>
<td>User cancelled transfer</td>
</tr>
</tbody>
</table>

Other error codes depend on the exchange library.

May display a fatal error message if socketP doesn’t contain a libraryRef value.

**Comments**

Applications must call this function when finished sending data or receiving data. It terminates the connection made with ExgConnect, ExgAccept, ExgPut, or ExgGet.

In the error parameter, pass any error that occurs during the application loop, including errors returned from other Exchange Manager functions. This ensures that the connection is shut down knowing that it failed rather than succeeded.

It’s especially important to check the result code from this function, since this will tell you if the transfer was successful. An errNone return value means that the item was delivered to the destination successfully. It does not mean that the user on the other end actually kept the data.

ExgDisconnect is used after sending and receiving. When receiving, the application can insert its creator ID into the goToCreator field in the socket structure and add other goto information in the goToParams field. After the application returns from the sysAppLaunchCmdExgReceiveData launch code, the exchange library may call ExgNotifyGoto, which launches the goToCreator application with the standard launch code sysAppLaunchCmdGoto.
IMPORTANT: Placing your creator ID in the goToCreator field is no longer a guarantee that you receive this launch code starting in Palm OS 4.0 because Palm OS 4.0 supports the sending of multiple objects at once. Thus, another application might overwrite the goToCreator field after your application has disconnected, making that application the recipient of the launch code.

Note that some exchange libraries wait to establish a connection until ExgDisconnect is called. The IR Library, for example, buffers the data that it receives and then waits until ExgDisconnect to actually send this data unless ExgConnect is called to establish a multi-object send connection.

Compatibility

Implemented only if 3.0 New Feature Set is present.

Prior to Palm OS release 4.0, the Exchange Manager always launched the goToCreator application, if one was provided, upon return from this function. If 4.0 New Feature Set is present, the Exchange Manager does not launch the goToCreator application. Exchange libraries that want the previous behavior must explicitly call ExgNotifyGoto.

See Also ExgReceive, ExgSend
**ExgDoDialog**

**Purpose**
Displays a dialog that allows users to accept or reject the receipt of data.

**Declared In**
ExgMgr.h

**Prototype**

```c
Boolean ExgDoDialog (ExgSocketType *socketP,
                     ExgDialogInfoType *infoP, Err *errP)
```

**Parameters**

- `socketP` A pointer to the socket structure (see `ExgSocketType`) identifying the connection. A value must be provided for the `libraryRef` field.
  
  Applications can obtain the socket structure from the `sysAppLaunchCmdExgAskUser` launch code parameter block.

- `infoP` A pointer to an `ExgDialogInfoType` structure (see the Comments section below).

- `errP` `errNone` if no error, or the error code if an error occurred. Currently, no errors are returned.

**Result**
Returns `true` if the user clicks the OK button on the dialog, or `false` otherwise.

**Comments**
This function displays the exchange dialog, which prompts the user to accept or reject incoming data.

By default, the Exchange Manager calls this function if the receiving application doesn’t handle the `sysAppLaunchCmdExgAskUser` launch code or if it returns `exgAskDialog` from the launch code handler. When the Exchange Manager calls `ExgDoDialog`, the dialog displays a message similar to “Do you want to accept ‘John Doe’ into Address Book?” and allows the user to accept or reject the data. If the user clicks OK, the data should be received as an unfiled record.
The Exchange Manager attempts to display a preview of the data in the exchange dialog to provide users with enough information to determine if they want to accept or reject the data. To display the preview data, it calls \texttt{ExgNotifyPreview}. Applications wishing to support preview mode should respond to the launch code \texttt{sysAppLaunchCmdExgPreview}. See the \texttt{ExgNotifyPreview} function’s description for more information.

Applications may also want to allow users to select a category in which to accept the incoming data. To do so, handle \texttt{sysAppLaunchCmdExgAskUser} to call \texttt{ExgDoDialog} directly and pass it a pointer to an \texttt{ExgDialogInfoType} structure. The \texttt{ExgDialogInfoType} structure is defined as follows:

\begin{verbatim}
typedef struct {
    UInt16     version;
    DmOpenRef  db;
    UInt16     categoryIndex;
} ExgDialogInfoType;
\end{verbatim}

- \texttt{version} Set this field to 0 to specify version 0 of this structure.
- \texttt{db} A pointer to an open database that defines the categories the dialog should display.
- \texttt{categoryIndex} The index of the category in which the user wants to file the incoming data.

If \texttt{db} is valid, the function extracts the category information from the specified database and displays it in a pop-up list. Upon return, the \texttt{categoryIndex} field contains the index of the category the user selected, or \texttt{dmUnfiledCategory} if the user did not select a category.

If the call to \texttt{ExgDoDialog} is successful, your application is responsible for retaining the value returned in \texttt{categoryIndex} and using it to file the incoming data as a record in that category.

One way to do this is to store the \texttt{categoryIndex} in the socket’s \texttt{appData} field (see \texttt{ExgSocketType}) and then extract it from the socket in your response to the launch code \texttt{sysAppLaunchCmdExgReceiveData}. For example:
if (cmd == sysAppLaunchCmdExgReceiveData) {
    UInt16 category =
        (ExgSocketPtr)cmdPBP->appData;
    /* other declarations */

    /* Receive the data, and create a new record using the received data. indexNew is the index of this record. */
    if (category != dmUnfiledCategory) {
        UInt16 attr;
        Err err;
        err = DmRecordInfo(dbP, indexNew, &attr, NULL, NULL);

        // Set the category to the one the user specified, and mark the record dirty.
        if ((attr & dmRecAttrCategoryMask) != category) {
            attr &= ~dmRecAttrCategoryMask;
            attr |= category | dmRecAttrDirty;
            err = DmSetRecordInfo(dbP, indexNew, &attr, NULL);
        }
    }
}

Some of the Palm OS built-in applications (Address Book, Memo, and ToDo) use this method of setting the category on data received through beaming. Refer to the example code for these applications provided in the SDK for a more complete example of how to use ExgDoDialog.

When you explicitly call ExgDoDialog, you must set the result field of the sysAppLaunchCmdExgAskUser launch code’s parameter block to either exgAskOk (upon success) or exgAskCancel (upon failure) to prevent the system from displaying the dialog a second time.

**Compatibility** Implemented only if 3.5 New Feature Set is present.
Preview mode display in the exchange dialog is implemented only if 4.0 New Feature Set is present.

**ExgGet**

**Purpose** Establishes a connection and requests an object from a remote device.

**Declared In** ExgMgr.h

**Prototype** Err ExgGet (ExgSocketType *socketP)

**Parameters**

- socketP A pointer to the socket structure (see ExgSocketType). Specify either a value for the libraryRef field or a URL in the name field. libraryRef should be 0 if the name field contains a URL. The target, type, or name fields should identify the data being requested.

**Result** Returns one of the following error codes:

- errNone Success
- exgErrBadLibrary Couldn’t find default exchange library
- exgErrUserCancel The user cancelled the operation
- exgMemError There is not enough free memory to perform the operation

Other error codes depend on the exchange library.

**Comments** Applications use this function to request data (initiate a send) from a remote device. Not all exchange libraries support this operation. Before calling this function, the application must initialize the socketP parameter. The socket should identify the exchange library to connect with by providing either a library reference number in the libraryRef field or a URL in the name field. The default exchange library registered for the URL’s scheme handles
the connection. The socket should also specify what data it is requesting by providing values for at least one of the target, name, and type fields. Specifying the data in the name field is the most common method.

To provide users with a choice of transport mechanisms, the application can provide a URL that begins with a question mark (?). The Exchange Manager displays a dialog with a list of all exchange libraries that respond to URLs of the specified type. If only one exchange library is registered for this URL scheme, no dialog is displayed.

If the library is not specified by either URL or library reference number, the Exchange Manager by default uses the IR Library; however, if the localMode flag is set, the Local Exchange Library is used instead.

Applications can use ExgGet to initiate a send from the Local Exchange Library. For more information, see “Sending and Receiving Locally” on page 32 of the Palm OS Programmer’s Companion, vol. II, Communications.

If no error is returned, applications should follow this call with one or more calls to ExgReceive, to receive the data, or ExgDisconnect, to disconnect.

**Compatibility**

Implemented only if [4.0 New Feature Set](#) is present.

**See Also**

ExgPut, ExgConnect
ExgGetDefaultApplication

Purpose
Retrieves the default application for the specified type of data or the default exchange library for URLs with the specified scheme.

Declared In
ExgMgr.h

Prototype
Err ExgGetDefaultApplication (UInt32 *creatorIDP, UInt16 id, const Char *dataTypeP)

Parameters
<- creatorIDP A pointer to the creator ID of the default application or default exchange library.
-> id The registry ID constant identifying the type of data in dataTypeP. See Registry ID Constants.
-> dataTypeP A pointer to a string that contains the type of data for which to retrieve the default application or library. If dataTypeP is a file extension, do not include the period (.). If it is a URL, do not include the colon (:).

Result
Returns errNone if a match was found or exgErrNoKnownTarget if there is no default application or library for this type of data.

Comments
You might use this function to see which application on this device will receive a particular type of data or to see which library on this device handles URLs of a particular scheme.

For example, to find out which application receives TXT files on this device, do the following:

UInt32 creatorID;
Err error;
error = ExgGetDefaultApplication(&creatorID, exgRegExtensionID, "TXT");
if (!error) {
    //creatorID contains default application.
}

To find out which exchange library handles URLs that use the beam prefix, do the following:
(UInt32 creatorID;
Err error;
error = ExgGetDefaultApplication(&creatorID,
exgRegSchemeID, exgBeamScheme);
if (!error) {
    //creatorID contains default library.
}

It’s possible to have several applications registered to receive the same type of data, but none of them is the default. When the Exchange Manager receives an object of that type, it selects an application to receive the data, and it selects that same application every time. The selected application effectively becomes the default for the data type even though it is not explicitly set as the default. If this is the case, the ExgGetDefaultApplication function returns the creator ID of this de-facto default application.

Compatibility
Implemented only if 4.0 New Feature Set is present.

See Also
ExgGetRegisteredApplications,
ExgGetRegisteredTypes, ExgRegisterDatatype,
ExgSetDefaultApplication

ExgGetRegisteredApplications

Purpose
Retrieves a list of all applications registered to receive data of a specified type.

Declared In
ExgMgr.h

Prototype
Err ExgGetRegisteredApplications
   (UInt32 **creatorIDsP, UInt32 *numAppsP,
    Char **namesP, Char **descriptionsP, Uint16 id,
    const Char *dataTypeP)

Parameters
creatorIDsP An array of the creator IDs of the applications registered to receive objects of this type. Pass NULL for this parameter if you only want to know how many applications are registered for this type.
The number of applications registered to receive objects of this type. This is the number of elements in the creatorIDsP array, the namesP array, and the descriptionsP array.

A packed list of strings, suitable for passing to SysFormPointerArrayToStrings, containing the names of the applications or libraries. Each string is no more than exgMaxTitleLen characters. Pass NULL for this parameter if you don’t want to retrieve it.

A packed list of strings, suitable for passing to SysFormPointerArrayToStrings, containing the descriptions of the applications or libraries. Descriptions are specified when the applications or libraries register for data. Each string is no more than exgMaxDescriptionLength characters. Pass NULL for this parameter if you don’t want to retrieve it.

The registry ID constant identifying the type of data in dataTypeP. See Registry ID Constants.

A pointer to a tab-delimited, null-terminated string listing the items to register. (Use \t for the tab character.) Each item in the string must be no more than exgMaxTypeLength characters. There can be no more than 16 types total.

Returns errNone upon success or exgMemError if the function cannot allocate space for the creator IDs, names, or descriptions.

IMPORTANT: This function allocates enough space for the creatorIDsP, namesP, and descriptionsP arrays as long as you do not pass NULL for the parameters. You are still responsible for freeing these arrays.
Comments  You might use this function to see which applications on this device can receive a particular type of data or to see which libraries on this device handle URLs of a particular scheme. You can also use it to build a list of choices from which the user can select a default application or default exchange library for a particular data type or URL scheme. For example, iMessenger uses this function to build a list of mailto handlers so that the user can choose one of them to be the default.

The Exchange Manager itself uses ExgGetRegisteredApplications to find exchange libraries when it is given a URL that begins with a question mark (?). It displays the returned list to the user in the Send With dialog.

Compatibility  Implemented only if 4.0 New Feature Set is present.

See Also  ExgGetDefaultApplication, ExgGetRegisteredTypes, ExgRegisterDatatype, ExgSetDefaultApplication

ExgGetRegisteredTypes

Purpose  Retrieve a list of all data types for which a registration exists.

Declared In  ExgMgr.h

Prototype  Err ExgGetRegisteredTypes (Char **dataTypesP, UInt32 *sizeP, UInt16 id)

Parameters  <- dataTypesP  A packed list of strings, suitable for passing to SysFormPointerArrayToStrings, containing a sorted list of data types for which a registration exists. Each string is no more than exgMaxTypeLength characters.

<- sizeP  The number of elements in the dataTypesP array.
-> id  The type of data to search for. For example, you can search for all registered creator IDs, all registered MIME types, and so on.

Result  Returns errNone upon success or exgMemError if the function cannot allocate space for the data types array.

**IMPORTANT:** This function allocates enough space for the dataTypeP array as long as you do not pass NULL for the parameter. You are still responsible for freeing this array.

Comments  This function could be used to create an application that allows users to choose the default application for each data type.

Compatibility  Implemented only if 4.0 New Feature Set is present.

See Also  ExgGetDefaultApplication, ExgGetRegisteredApplications, ExgRegisterDatatype, ExgSetDefaultApplication

### ExgGetTargetApplication

Purpose  Retrieves the application that should receive a specific message. This function does not search for libraries.

Declared In  ExgMgr.h

Prototype  Err ExgGetTargetApplication
            (ExgSocketType *socketP, Boolean unwrap, UInt32 *creatorIDP, Char *descriptionP, UInt32 descriptionSize)

Parameters  -> socketP  A pointer to the socket structure (see ExgSocketType). The structure should contain values for the target, type, or name fields.
Exchange Manager
Exchange Manager Functions

-> unwrap
If true, only an application that registered to receive the data type with the exgUnwrap flag set should be the target application. If false, the target application should be an application that registered with the exgUnwrap flag clear.

<- creatorIDP
The creator ID of the application that should receive this object.

<-> descriptionP
The application’s description from the registry, if any.

-> descriptionSize
The size of the descriptionP buffer.

Result
Returns one of the following error codes:

errNone
Success

exgErrTargetMissing
The target field contains a creator ID, but the application with that creator ID does not exist

exgErrNoKnownTarget
No application is registered to receive the data type

Comments
The Exchange Manager uses this function to determine which application should be launched to receive incoming data. Applications and libraries may call this function as well.

ExgGetTargetApplication determines the target application by doing the following:

- If the socketP->target field contains a creator ID, the Exchange Manager searches the registry to see if an application is registered for that creator ID as the default application. If the registry does not contain an entry for the creator ID, it checks to see if the application identified by the creator ID is installed on this device. If an application is found for the target, that is the application returned.

- If the socketP->type field contains a MIME type, the Exchange Manager searches the registry for an application
registered to receive objects of that type. If one is found, that is the application returned.

- If the `socketP->name` field contains a period (.), the portion after the last period is taken to be the file extension. The Exchange Manager searches the registry for an application registered to receive a file with the specified extension. If one is found, that is the application returned. If not, `exgErrNoKnownTarget` is returned.

If more than one application is registered for the target, type, or file extension, this function returns the one that is registered as the default. If no application is registered as the default, then a specific application is chosen. The Exchange Manager chooses this same application each time. That is, each time a file with a TXT extension is sent with no target or MIME type specified, the `ExgGetTargetApplication` returns the same application to handle the receipt.

Set the `unwrap` parameter to `true` if the object was sent as part of another object, such as a vStock object that was sent as an attachment to an e-mail message. In this case, the Exchange Manager searches for an application that registered to receive the target, the type, or the file extension of the vStock object with the `exgUnwrap` flag set. If an application is found, the vStock object is delivered, and the exchange library should discard the object that contained it (the e-mail message). If there is no application registered to receive the data with the `exgUnwrap` flag set, this function returns `exgErrNoKnownTarget`. In this case, the exchange library should call `ExgNotifyReceive` again passing the entire e-mail message instead of just the vStock attachment.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.

**See Also**

`ExgSetDefaultApplication`, `ExgNotifyPreview`, `ExgNotifyReceive`, `ExgRegisterDatatype`
ExgNotifyGoto

Purpose
Launches the target application using `sysAppLaunchCmdGoto`.

Declared In
`ExgMgr.h`

Prototype
```c
Err ExgNotifyGoto (ExgSocketType *socketP, Uint16 flags)
```

Parameters
- `socketP`: A socket identifying the object to deliver (see `ExgSocketType`). The `goToCreator` field contains the application to be launched, and the `goToParams` field contains data for the launch code's parameter block.

- `flags`: Not currently used. Pass 0 for this parameter.

Result
Returns one of the following error codes:

- `errNone`: Success or the `goToCreator` field is empty
- `dmErr...`: (one of the data manager error codes) The specified application could not be found
- `memErrNotEnoughSpace`: Not enough memory available to create the launch code’s parameter block

Comments
Exchange libraries call this function if they want to support immediate display of the received object. Applications do not call this function.

Most exchange libraries should call `ExgNotifyGoto` after the return from `ExgNotifyReceive` so that the user can inspect the newly received data. If the exchange library is most often used by a single application that does not require the launch code, this call to `ExgNotifyGoto` can be skipped. For example, the SMS Library does not call `ExgNotifyGoto`. SMS messages are received by the SMS Messenger application, which does not launch upon receiving data.


**Exchange Manager**

**Exchange Manager Functions**

ExgNotifyGoto only launches an application if one is specified in the gotoCreator field and the noGoTo parameter is false. If a gotoCreator is not specified, it is not considered an error. This gives the application a way to override the default behavior.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.

**See Also**

ExgNotifyReceive, ExgDisconnect

---

**ExgNotifyPreview**

**Purpose**

Displays a preview in the exchange dialog of the data to be received.

**Declared In**

ExgMgr.h

**Prototype**

Err ExgNotifyPreview (ExgPreviewInfoType *infoP)

**Parameters**

<-> infoP An ExgPreviewInfoType structure containing information about the preview operation.

**Result**

Returns one of the following error codes:

- errNone Success
- exgErrNotSupported The exchange library doesn’t support preview mode
- exgErrNoKnownTarget There is no application registered to receive the type of object

Other error codes depend on the application.

**Comments**

This function performs the preview operation specified in the op field of the infoP parameter. The ExgDoDialog function calls this function to show a data preview in the exchange dialog. Exchange libraries might want to call this function in certain circumstances. An application rarely calls this function, but it may do so if it displays its own dialog in response to the launch code sysAppLaunchCmdExgAskUser.
ExgNotifyPreview uses `ExgGetTargetApplication` to determine the appropriate target application for this data and then launches that application with the launch code `sysAppLaunchCmdExgPreview`, passing `infoP` as the parameter block. The application responds to this launch code by accepting the connection, receiving the data from the exchange library, and depending on the operation requested, drawing the data into the `infoP->bounds` rectangle or returning it in the `infoP->string` field, and then disconnecting. The `ExgDoDialog` function uses the returned information to draw the preview portion of the dialog.

If the preview data is a string, the `ExgNotifyPreview` provides a series of fallback strings that are used if the exchange library doesn’t support preview or the application doesn’t respond to the launch code. If the application fails to return a string, this function provides one of the following:

- the data’s description from `socketP->description`
- the filename in `socketP->name`
- the target application’s description as stored in the exchange registry
- the MIME type in `socketP->type`
- the file extension in `socketP->name`

### Compatibility
Implemented only if 4.0 New Feature Set is present.

### See Also
`ExgNotifyReceive`, `ExgDisconnect`

---

**ExgNotifyReceive**

**Purpose**
Delivers an object to the appropriate application using the registry.

**Declared In**
`ExgMgr.h`

**Prototype**
```
Err ExgNotifyReceive (ExgSocketType *socketP, UInt16 flags)
```

**Parameters**
- `socketP` A pointer to the socket structure (see `ExgSocketType`).
-> flags  A bit field. Pass 0 or a combination of the following constants (OR the constants together to specify more than one):

exgUnwrap  The object being delivered should only be handled by an application that registered to receive it with the exgUnwrap flag set.

exgNoAsk  Do not ask the user to confirm receipt of data. If this constant is passed, the target application does not receive the sysAppLaunchCmdExgAskUser launch code, and the Exchange Manager does not call ExgDoDialog to display the user confirmation dialog.

exgGet  Specifies that this is a request for the application to send data rather than to receive data.

Result  Returns one of the following error codes:

errNone  Success

exgErrTargetMissing  The target field contains a creator ID, but the application with that creator ID does not exist

exgErrNoKnownTarget  No application is registered to receive the data type

exgErrUserCancel  The user cancelled the operation

Other error codes depend on the application that is launched.

Comments  Exchange libraries call this function to initiate a receive operation on the receiving device. Applications do not call this function.

The ExgNotifyReceive function uses ExgGetTargetApplication to determine which application
should receive the data, then sends that application the appropriate launch codes.

If the flags parameter is 0, a receive operation is assumed. The ExgNotifyReceive function does the following:

1. It sends the application the sysAppLaunchCmdExgAskUser launch code.
2. If the application returns exgAskDialog or does not respond to the launch code, it calls ExgDoDialog, which sends the application the sysAppLaunchCmdExgPreview launch code to have the application receive preview data for the dialog.
3. It sends the application the sysAppLaunchCmdExgReceiveData launch code to tell the application to receive the data.

If the flags field contains the exgNoAsk flag, the first and second steps are skipped.

If the flags field contains exgGet, this function is a request for data to send to the remote device, not a request to receive data from the remote device. In this case, ExgNotifyReceive launches the target application with the sysAppLaunchCmdExgGetData launch code.

If the flags field has the exgUnwrap bit set, it means that the object to be received was sent as part of another object, and it should only be sent to an application that registered to receive it with the exgUnwrap flag set. For example, if the exchange library receives an e-mail message with an attached vStock object, the exchange library may call ExgNotifyReceive with the exgUnwrap flag set and a socket that describes the vStock data type to see if there is an application that registered to receive it directly. If no application is registered to receive vStock objects with the exgUnwrap flag set, ExgNotifyReceive returns exgErrNoKnownTarget. The exchange library should then call ExgNotifyReceive again, but this time without the exgUnwrap flag and with a socket that describes the e-mail message data type. This second call sends the object to the application registered to receive the e-mail message rather than its vStock attachment. That application may extract the vStock attachment from the message and use the Local Exchange Library to send it to an application registered to receive vStock objects normally (without the exgUnwrap flag).
**Compatibility**

Implemented only if [3.0 New Feature Set](#) is present.

ExgNotifyReceive was a system use only function until the release of Palm OS 4.0.

If the [4.0 New Feature Set](#) is not present, the flags parameter is not supported, so libraries cannot suppress the exchange dialog, send objects with attachments, or perform a get operation. These features are all added in the [4.0 New Feature Set](#). Also, if the 4.0 new feature set is not present, this function performs the equivalent of ExgNotifyGoto after the application has returned from receiving data. Exchange libraries wishing to support this functionality should call ExgNotifyGoto immediately after calling ExgNotifyReceive.

**See Also**

ExgNotifyPreview

---

**ExgPut**

**Purpose**

Initiates the transfer of data to the destination device.

**Declared In**

ExgMgr.h

**Prototype**

Err ExgPut (ExgSocketType *socketP)

**Parameters**

- socketP

  Pointer to the socket structure (see [ExgSocketType](#)). Specify either a value for the libraryRef field or a URL in the name field. libraryRef should be 0 if the name field contains a URL. The structure should also contain a value for the target, type, or name field.

**Result**

Returns one of the following error codes:

- errNone: Success
- exgErrBadLibrary: Couldn’t find default exchange library
Other error codes depend on the exchange library.

**Comments**

Applications call this function to start a send operation.

If the connection does not already exist, this function establishes one. You must create and initialize an `ExgSocketType` structure containing information about the data to send and the destination application. All unused fields in the structure must be set to 0.

If no error is returned, this call must be followed by `ExgSend`, to begin sending data, or `ExgDisconnect`, to disconnect. You may need to call `ExgSend` multiple times to send all the data.

The socket’s `libraryRef` field or the `name` field must identify the library that performs the transfer. The `libraryRef` field identifies the exchange library by its library reference number. The `name` field identifies the library by URL. The socket should also specify what data is being sent by providing values for at least one of the `target`, `name`, and `type` fields. Use of the `name` field is the most common method.

To provide users with a choice of transport mechanisms, the application can provide a URL that begins with a question mark (?). The Exchange Manager displays a dialog with a list of all exchange libraries that respond to URLs of the specified type. If only one exchange library is registered for this URL scheme, no dialog is displayed.

For example, many applications on Palm OS 4.0 or higher support a Send command. This command generates a URL with the prefix `exgSendPrefix` (see Predefined URL Prefixes). The Exchange Manager displays a dialog containing a list of libraries registered for that URL scheme. The user selects an exchange library, and that library’s `ExgLibSend` function is called.

If the library is not specified by either URL or library reference number, the Exchange Manager by default uses the IR Library;
however, if the localMode flag is set, the Local Exchange Library is used instead.

**Compatibility**

Implemented only if [3.0 New Feature Set](#) is present.

Support for identifying exchange libraries by URL is implemented only if [4.0 New Feature Set](#) is present.

**See Also**

ExgDisconnect, ExgSend, ExgConnect

### ExgReceive

**Purpose**

Receives data from a remote device.

**Declared In**

ExgMgr.h

**Prototype**

```
UInt32 ExgReceive (ExgSocketType *socketP, void *bufP, UInt32 bufLen, Err *err)
```

**Parameters**

- `socketP` A pointer to the socket structure (see ExgSocketType).
- `bufP` A pointer to the buffer in which to receive the data.
- `bufLen` The number of bytes to receive.
- `err` A pointer to an error code result.

**Result**

Returns the number of bytes actually received. A zero result indicates the end of the transmission.

An error code is returned in the address indicated by `err`. The error code exgErrUserCancel is returned if the user cancels the operation. The error code exgErrNotSupported is returned if the application calls this function during a preview and the exchange library does not have any more data available or does not support preview.

May display a fatal error message if the library reference number is not provided in `socketP`. 
Comments
Applications call this function in the following circumstances:

- In response to the `sysAppLaunchCmdExgReceiveData` launch code, following a successful call to `ExgAccept`.
- In response to the `sysAppLaunchCmdExgPreview` launch code, following a successful call to `ExgAccept`.
- To receive requested data following a successful call to `ExgGet`.

After receiving the data, applications call `ExgDisconnect` to terminate the connection.

This function blocks the application until the end of the transmission or until the requested number of bytes has been received. However, exchange libraries can provide their own user interface that is shown during this call, is updated as necessary, and allows the user to cancel the operation in progress.

Compatibility
Implemented only if 3.0 New Feature Set is present. Preview mode and `ExgGet` are only supported if 4.0 New Feature Set is present.

See Also `ExgNotifyReceive`  

**ExgRegisterDatatype**

Purpose
Registers an application to receive a specific type of data, or registers an exchange library to handle specific URL schemes.

Declared In `ExgMgr.h`

Prototype
```c
Err ExgRegisterDatatype (UInt32 creatorID, 
UInt16 id, const Char *dataTypesP, 
const Char *descriptionsP, UInt16 flags)
```

Parameters
- `creatorID`  
The creator ID of the registering application or exchange library.
- `id`  
A registry ID constant identifying the type of the items being registered. See `Registry ID Constants`.  

-> dataTypesP
Pointer to a tab-delimited, null-terminated string listing the items to register. (Use "\t" for the tab character.) To unregister, pass a NULL value. Each item in the string must be no more than exgMaxTypeLength characters. There can be no more than 16 types total.

NOTE: If specifying file extensions, do not include the period (.) that precedes the extension. If specifying URL prefixes, do not include the colon (:) at the end of the prefix.

-> descriptionsP
Pointer to a tab-delimited, null-terminated string that lists descriptions for the items in the dataTypesP parameter. (Use "\t" for the tab character.) Each description must be no longer than exgMaxDescriptionLength. Pass NULL to leave out the descriptions.

There must either be one description for all types or the number of descriptions must match the number of types.

The descriptions are used in dialogs displayed by Exchange Manager to identify applications or libraries.
-> flags

A bit field specifying registration options. Currently, only one bit is used: the unwrap bit. Pass the exgUnwrap constant to specify that the application is registering to receive objects of this type directly if the object is sent as part of another object. For example, if a vStock object is sent as an attachment to an email message, the Exchange Manager should send the vStock object to this application directly rather than sending the message to the email application.

Result

Returns errNone if successful, exgMemError if there is not enough memory to save the registration info, or one of the data manager error codes (dmErr...).

Comments

Both applications and exchange libraries use this function to register with the Exchange Manager to receive certain types of data.

Applications must register with the Exchange Manager to receive data objects that do not specifically target that application using the creator ID in the target field.

Exchange libraries register to receive data with certain URL schemes. If an exchange library is not registered to receive URLs, it only handles the receipt and sending of data if its library reference number is explicitly specified in the ExgSocketType structure. Otherwise, the IR Library handles all incoming data for which a library could not be found.

Both applications and libraries should register to receive data as soon as possible after they are installed and as soon as possible after a hard reset. For example, applications can call ExgRegisterDatatype in response to the sysAppLaunchCmdSyncNotify launch code, which they receive immediately after install. Exchange libraries implemented as applications can also use this strategy. Exchange libraries implemented as shared libraries should call ExgRegisterDatatype in their startup functions.

Make only one call to ExgRegisterDatatype per registry type. If you want to register to receive multiple items, use a tab character...
(\t) to separate the items. If you were to, for example, make one call to register for the DOC file extension and one call to register for the TXT extension, the second call overwrites the first. However, if you want to register with the `exgUnwrap` flag set, make one call without the `exgUnwrap` flag and one call with the `exgUnwrap` flag set. The application registered with the `exgUnwrap` flag set is stored in a different part of the registry.

Specify `exgRegExtensionID` to register to receive data that has a filename with a particular extension. For example, if your application wants to receive files with a TXT extension, it could register like this:

```c
ExgRegisterDatatype(myCreator, exgRegExtensionID, "TXT", NULL, 0);
```

If the application wants to receive files with a TXT extension or with a DOC extension, it could register like this:

```c
ExgRegisterDatatype(myCreator, exgRegExtensionID, "TXT\tDOC", NULL, 0);
```

Specify `exgRegTypeID` to register to receive data with a specific MIME type. For example, if your application wants to receive “setext” text files, it could register like this:

```c
ExgRegisterDatatype(myCreator, exgRegTypeID, "text/x-setext", NULL, 0);
```

Specify `exgRegCreatorID` to register to receive data targeted for a particular creator ID. For example, if your application wants to handle all data intended for the ToDo application, it could register like this:

```c
Char toDoCreatorStr[5];
MemMove(toDoCreatorStr, sysFileCToDo, 4);
ptoDoCreatorStr[4] = chrNull;
ExgRegisterDatatype(myCreator, exgRegCreatorID, toDoCreatorStr, NULL, 0);
```

**NOTE:** To override one application’s receipt of data, you need to also set your application as the default for this creator ID. See `ExgSetDefaultApplication`. 
Most exchange libraries will want to register for a unique URL scheme that identifies only that library, plus they should register for a more general scheme, such as the send scheme (exgSendScheme), which causes the library to be listed in the Send With dialog when the user performs the Send command. The registry ID constant for URL prefixes is exgRegSchemeID.

```c
ExgRegisterDatatype(myLibCreator, 
    exgRegSchemeID, myScheme "\t" exgSendScheme, 
    NULL, 0);
```

Registrations are active until a hard reset or until the application or library is removed. The registration information is preserved across a soft reset. When an application is removed, its registry information is also automatically removed from the registry, so there is not normally a need to unregister. If you want to unregister, you can call ExgRegisterDatatype with a NULL value for the dataTypesP parameter.

Multiple applications can be registered to receive the same type of data. If this is the case, the application that is registered as the default (using ExgSetDefaultApplication) is the one that receives the data unless the exchange socket explicitly specifies another application should receive it. If there is no default specified, the Exchange Manager determines a default.

Multiple libraries may also be registered to receive the same type of URL. In this case, if the URL begins with a question mark (?), the Exchange Manager displays a dialog so that the user can select which exchange library to use. If the URL does not begin with a question mark, the exchange library registered as the default is used. If there is no default specified, the Exchange Manager determines a default.

**Compatibility**

Implemented only if 4.0 New Feature Set is present. ExgRegisterDatatype replaces the ExgRegisterData function.

**See Also**

ExqRegisterData, ExgGetTargetApplication, ExgPut, ExgGetDefaultApplication, ExgGetRegisteredApplications, ExgGetRegisteredTypes
**ExgRegisterData**

**Purpose**: Registers an application to receive a specific type of data. This function is deprecated and replaced with `ExgRegisterDatatype`.

**Declared In**: ExgMgr.h

**Prototype**: `Err ExgRegisterData (UInt32 creatorID, UInt16 id, const Char *dataTypesP)`

**Parameters**
- `creatorID` Creator ID of the registering application.
- `id` Registry ID identifying the type of the items being registered. Specify `exgRegExtensionID` or `exgRegTypeID`.
- `dataTypesP` Pointer to a tab-delimited, null-terminated string listing the items to register. (Use \t for the tab character.) These include file extensions or MIME types. To unregister, pass a NULL value.

**Result**: Returns `errNone` if successful, otherwise, one of the data manager error codes (`dmErr...`).

**Comments**: Applications that wish to receive data from anything other than another Palm Powered™ handheld running the same application must use this function to register for the kinds of data they can receive. Call this function when your application is loaded on the device.

**Compatibility**: This function corresponds to the Palm OS 3.5 version of `ExgRegisterDatatype`. It is implemented only if 3.0 New Feature Set is present.
Exchange Manager
Exchange Manager Functions

ExgRequest

Purpose Requests some data from an exchange library or an application using a URL.

Declared In ExgMgr.h

Prototype Err ExgRequest (ExgSocketType *socketP)

Parameters -> socketP Pointer to the socket structure (see ExgSocketType). Specify a URL in the name field and a libraryRef of 0.

Result Returns one of the following error codes:

errNone Success
exgErrBadLibrary Couldn’t find default exchange library
exgErrNotEnoughPower The device does not have enough power to perform the operation
sysErrLibNotFound Couldn’t find library or application to respond to URL

Other error codes depend on the exchange library or application.

Comments The ExgRequest function is similar to ExgGet in that both are used to request data. The difference is that the application that calls ExgGet is always the application that receives the data. When you call ExgRequest, the application that receives the data is the application that is registered to receive it. For example, using ExgRequest, it is possible for one application to use the Exchange Manager to retrieve a vCard using any supported transport mechanism and have that data sent directly to the Address Book application instead of to the calling application.

The socketP passed to this function identifies the exchange library using a URL in the name field. The application must know beforehand the proper URL prefix for the exchange library with which it
wants to connect. See Predefined URL Prefixes for a list of URL prefixes that the Exchange Manager provides.

If the provided URL begins with a question mark (?) and there are several exchange libraries registered for the specified URL scheme, the Exchange Manager displays a dialog from which the user selects the appropriate transport mechanism.

If the Exchange Manager cannot find a library that is registered for the specified URL, it assumes that an application is registered to receive the URL, and it launches that application with the sysAppLaunchCmdGoToURL launch code.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also ExgGet, ExgNotifyReceive

ExgSend

Purpose Sends data to the destination device.

Declared In ExgMgr.h

Prototype UInt32 ExgSend (ExgSocketType *socketP, const void *bufP, UInt32 bufLen, Err * err)

Parameters

- > socketP A pointer to the socket structure (see ExgSocketType). A value must be provided for the libraryRef field. The structure should also contain values for the target, type, or name fields.
- > bufP A pointer to the data to send.
- > bufLen The number of bytes to send.
- <- err A pointer to an error code result.

Result Returns the number of bytes actually sent, normally the same number as specified in bufLen. An error code is returned in the address indicated by err. The error code exgErrUserCancel is returned if the user cancels the operation.
May display a fatal error message if the socketP parameter does not contain a value for the libraryRef field.

**Comments**

Call this function one or more times to send all the data, following a successful call to `ExgPut`. After sending the data, call `ExgDisconnect` to terminate the connection.

The exchange library may break large amounts of data into multiple packets or assemble small send commands together into larger packets, but the application will not be aware of these transport level details.

This function blocks the application until all the data is sent. However, the exchange library may provide its own user interface that is updated as necessary and allows the user to cancel the operation in progress.

**Compatibility**

Implemented only if 3.0 New Feature Set is present.

**See Also**

`ExgReceive`, `ExgGet`

---

**ExgSetDefaultApplication**

**Purpose**

Sets the application that receives a specified type of data by default. This function also sets the default exchange library that handles particular URL schemes.

**Declared In**

`ExgMgr.h`

**Prototype**

```c
Err ExgSetDefaultApplication (UInt32 creatorID, UInt16 id, const Char *dataTypeP)
```

**Parameters**

- `creatorID` The creator ID of the application or library that should become the default for this type of data.
- `id` A registry ID constant identifying the type of data in `dataTypeP`. See `Registry ID Constants`.
- `dataTypeP` A pointer to a null-terminated string containing the desired type of data.
NOTE: If specifying a file extension, do not include the period (.) that precedes the extension. If specifying a URL prefix, do not include the colon (:) at the end of the prefix.

Result

Returns errNone upon success or exgErrNoKnownTarget if the specified application is not registered to receive the specified data type.

Comments

This function sets the default application that receives data of a certain type when no target is specified and the default exchange library that handles URLs with a certain prefix if no library reference number is specified.

Palm™ strongly recommends that applications allow the user to determine which application should become the default recipient for a data type. To do so, an application can use ExgGetRegisteredApplications to get the list of applications registered for the same type of data as it is, and then display a dialog listing those applications and allow the user to select it. Then it should call ExgSetDefaultApplication with the user-specified default.

If you call ExgSetDefaultApplication with an application or library that is already the default, this function has no effect.

An application can become the default for its own creator ID even if it has not specifically registered to receive its own creator ID. That is, suppose several applications are registered to receive objects targeted for the ToDo application’s creator ID. The ToDo application itself is not registered for its own creator ID, as it is not necessary to do so. However, an application can use code like the following to set the ToDo application as the default for its own creator ID.

```c
Char toDoCreatorStr[5];
MemMove(toDoCreatorStr, sysFileCToDo, 4);
toDoCreatorStr[4] = chrNull;
ExgSetDefaultApplication(sysFileCToDo,
exgRegCreatorID, toDoCreatorStr);
```
Exchange Manager
Application-Defined Functions

Compatibility
Implemented only if 4.0 New Feature Set is present.

See Also
ExgGetDefaultApplication, ExgRegisterDatatype

Application-Defined Functions

ExgDBDeleteProcPtr

Purpose
Handles the case where a database with an identical name already exists on the device.

Declared In
ExgMgr.h

Prototype
Boolean (*ExgDBDeleteProcPtr) (const char* nameP, UInt16 version, UInt16 cardNo, LocalID dbID, void* userDataP)

Parameters
- > nameP A pointer to the name of the identical database.
- > version The version of the identical database.
- > cardNo The card number of the identical database.
- > dbID The database ID of the identical database.
- > userDataP The userDataP parameter you passed to ExgDBRead. If used, this parameter contains any application-specific data you find necessary. If the ExgDBReadProcPtr function is implemented using Exchange Manager calls, this often contains the ExgSocketType structure.

Result
Return true to have the ExgDBRead function continue to read the database. Use this return value if you have deleted or moved the existing database or if you want the database to be overwritten. Return false to have ExgDBRead exit without reading the database.
Comments  This function is called if the Data Manager can’t create the incoming
database because a database with the same name already exists. You
should delete the existing database or take some other action, such
as changing the database name. It is appropriate to prompt the user
before choosing to delete or move the database.

ExgDBReadProcPtr

Purpose  Reads in the database and pass it to ExgDBRead.

Declared In  ExgMgr.h

Prototype  Err (*ExgDBReadProcPtr) (void* dataP,
UInt32* sizeP, void* userDataP)

Parameters  
<- dataP  A pointer to a buffer where this function should
place the database data. This buffer is allocated
in the dynamic heap by ExgDBRead; you don’t
need to use DmWrite when filling it.

<-> sizeP  The size of dataP. This value is set by
ExgDBRead to the number of bytes it expects to
receive in dataP. You must set this value to the
number of bytes you return in dataP (if it’s not
the same).

-> userDataP  The userDataP parameter you passed to
ExgDBRead. Pass the ExgSocketType
structure if you implement this function using
Exchange Manager calls.

Result  Return an error number, or errNone if there is no error. If this
function returns an error, ExgDBRead deletes the database it was
creating, cleans up any memory it allocated, then exits, returning
the error passed back from this function.

Comments  ExgDBRead is commonly used to receive a database from a beam or
from some other transport mechanism. In this case, an appropriate
implementation of this callback function is to call ExgReceive as
shown here:
Err MyReadDBProc (void *dataP, UInt32 *sizeP,
    void *userDataP)
{
    Err err = errNone;
    //userDataP contains ExgSocketType pointer.
    *sizeP =
        ExgReceive((ExgSocketType *)userDataP,
            dataP, *sizeP, &err);
    return err;
}

ExgDBWriteProcPtr

Purpose
Writes out the database.

Declared In
ExgMgr.h

Prototype
Err (*ExgDBWriteProcPtr) (const void* dataP,
    UInt32* sizeP, void* userDataP)

Parameters
- > dataP A pointer to a buffer containing the database data, placed there by ExgDBWrite.

  <-> sizeP The number of bytes placed in dataP by ExgDBWrite. If you were unable to write out or send all of the data in this chunk, on exit, set sizeP to the number of bytes you did write.

- > userDataP The userDataP parameter you passed to ExgDBWrite. You can use it for application-specific data. Pass the ExgSocketType structure if you implement this function using Exchange Manager calls.

Result
Return an error number, or errNone if there is no error. If this function returns an error, ExgDBWrite closes the database it was reading, cleans up any memory it allocated, then exits, returning the error passed back from this function.
Comments  ExgDBWrite is commonly used to write a database that is going to be beamed to another device (or sent through some other transport mechanism). In this case, an appropriate implementation of this callback function is to call ExgSend as shown here:

```c
Err MyWriteDBProc (void *dataP, UInt32 *sizeP, void *userDataP)
{
    Err err = errNone;
    //userDataP contains ExgSocketType pointer.
    *sizeP =
        ExgSend((ExgSocketType *)userDataP, dataP, *sizeP, &err);
    return err;
}
```
Exchange Library

The Exchange Library API described in this chapter and declared in ExgLib.h specifies a minimal set of functions that all exchange libraries must implement. This chapter is directed towards developers who use or create exchange libraries. Developers creating an exchange library should also read the Exchange Libraries chapter of the Palm OS Programmer’s Companion, vol. II, Communications.

Exchange Library Functions

**ExgLibAccept**

**Purpose**
Accept an incoming connection.

**Declared In**
ExgLib.h

**Prototype**
Err ExgLibAccept(UInt16 libRefnum, ExgSocketType *exgSocketP)

**Parameters**
- `libRefnum` Reference number of this exchange library.
- `exgSocketP` A pointer to the socket structure (see ExgSocketType).

**Result**
Returns errNone if no error. exgErrNotSupported is returned if a preview is in progress and the exchange library does not support preview. Other error codes are defined by each exchange library.

**Comments**
The Exchange Manager’s ExgAccept function simply calls ExgLibAccept in the exchange library identified by the ExgSocketType structure passed to ExgAccept. An application calls the Exchange Manager’s ExgAccept function when:
• The application wants to initiate a connection to receive data, which it does in response to `sysAppLaunchCmdExgReceiveData`.

• The application wants to initiate a connection to receive a preview of the data, which it does in response to `sysAppLaunchCmdExgAskUser`.

Any implementation of `ExgLibAccept` should update any progress dialogs to indicate that data is being accepted (or received) into an application.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**

`ExgLibPut`

### ExgLibClose

**Purpose**
Library-specific. Although this function is not called by the Exchange Manager, all shared libraries normally implement it.

**Declared In**
`ExgLib.h`

**Prototype**
```c
Err ExgLibClose (UInt16 libRefnum)
```

**Parameters**

- `libRefnum` Reference number of this exchange library.

**Comments**
Exchange libraries are free to implement this function for internal or external use. The Exchange Manager does not call it.
ExgLibConnect

**Purpose**
Open a connection in preparation for sending or receiving objects.

**Declared In**
ExgLib.h

**Prototype**
```c
Err ExgLibConnect(UInt16 libRefNum,
                  ExgSocketType *exgSocketP)
```

**Parameters**
- `libRefNum` Reference number of this exchange library.
- `exgSocketP` A pointer to an `ExgSocketType` structure identifying the socket through which objects will be sent or received.

**Result**
Returns `errNone` if no error. If `ExgLibConnect` is not supported by this library, this function returns `exgErrNotSupported`; if its use is optional, `errNone` is returned. Other error codes are defined by each exchange library.

**Comments**
The Exchange Manager may call this function to initiate a connection for sending multiple objects or for performing two-way communications. Some exchange libraries support sending multiple objects but do not support this call. See “Sending Multiple Objects” on page 17 of Palm OS Programmer’s Companion, vol. II, Communications for more information.

Not all exchange libraries support this operation. In this case, the first call to `ExgLibPut` must clean up after itself before returning an error. The exchange library should not expect to get an `ExgLibDisconnect` call.

If `ExgLibConnect` is supported and an application calls `ExgConnect`, the exchange library should delay any cleanup until `ExgLibConnect` returns an error, in which case it should clean up after itself. The library can expect to get an `ExgLibDisconnect` call if it returns `errNone` from `ExgLibConnect`. If the application does not call `ExgConnect`, the first call to `ExgLibPut` must clean up after itself before returning an error and the exchange library should not expect to get an `ExgLibDisconnect` call.
The Exchange Manager’s ExgConnect function calls ExgLibConnect in the exchange library identified by the ExgSocketType structure passed to ExgConnect. If ExgLibConnect is implemented to return exgErrNotEnoughPower, the Exchange Manager puts up an alert, so there is no need for the exchange library to do so. Other error codes are not treated specially by the Exchange Manager; the exchange library must put up its own alerts when appropriate.

The exchange library may prompt the user for addressing information.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.

### ExgLibControl

**Purpose**
Supply information about the exchange library.

**Declared In**
ExgLib.h

**Prototype**

```
Err ExgLibControl(UInt16 libRefNum, UInt16 op, 
                  void *valueP, UInt16 *valueLenP)
```

**Parameters**

- **-> libRefNum**
  Reference number of this exchange library.
- **-> op**
  The operation to perform.
- **<-> valueP**
  An operation specific parameter. See comments below.
- **<-> valueLenP**
  An operation specific parameter. See comments below.

**Result**

Returns errNone if no error. exgErrNotSupported is returned if the specified operation is not supported by the exchange library. exgErrBadParam is returned if the parameters (valueP and valueLenP) are not appropriate for the operation. Additional error codes are defined by each exchange library.

**Comments**

ExgLibControl is a general purpose function that performs various minor operations based upon a selector. ExgMgr.h defines
three selectors: \texttt{exgLibCtlGetTitle}, which should be supported by all exchange libraries, \texttt{exgLibCtlGetVersion}, and \texttt{exgLibCtlGetPreview}. Additional library-specific selectors should be numbered starting at \texttt{exgLibCtlSpecificOp} (0x8000).

Upon receiving \texttt{exgLibCtlGetTitle}, the exchange library must return a library title suitable for use in Exchange Manager dialogs. Exchange libraries that are built as applications should generally return their 'tAIN' resource as their title. Be sure that the title returned through \texttt{valueP} honors the maximum length specified in \texttt{valueLenP}.

An exchange library that implements the Palm OS 4.0 version of the Exchange Library API needn't do anything special upon receiving \texttt{exgLibCtlGetVersion}; it should simply return \texttt{exgErrNotSupported}. Otherwise, set \texttt{valueP} to a two-byte value indicating the API version number and return \texttt{errNone}. Note that a version number of zero corresponds to the Palm OS 4.0 version of the Exchange Library API.

The \texttt{exgLibCtlGetPreview} operation is used by the Exchange Manager to determine whether a given exchange library supports preview. The Exchange Manager assumes that an exchange library supports preview if the exchange library doesn't implement this operation. To indicate that a library does not support preview, set \texttt{valueP} to \texttt{false} and return \texttt{errNone}.

The Exchange Manager's \texttt{ExgControl} function simply calls \texttt{ExgLibControl} in the exchange library identified by the \texttt{ExgSocketType} structure passed to \texttt{ExgControl}.

**Compatibility** Implemented only if 4.0 New Feature Set is present.
ExgLibDisconnect

**Purpose**
Disconnect a connection made with ExgLibConnect, ExgLibAccept, ExgLibPut, or ExgLibGet.

**Declared In**
ExgLib.h

**Prototype**
Err ExgLibDisconnect(UInt16 libRefnum, ExgSocketType *exgSocketP, Err error)

**Parameters**
- **libRefnum**
  Reference number of this exchange library.
- **exgSocketP**
  A pointer to an ExgSocketType structure identifying the socket connection to be disconnected.
- **error**
  The current error state. Used to indicate why the connection is being broken, for example, user cancel or out of memory.

**Result**
Typically the same error code passed in. However, this function may return an error even if errNone is passed in.

**Comments**
ExgLibDisconnect may be used to finish reading the data during a preview; in this case, the connection, if any, is not shut down.

Applications call ExgLibDisconnect when all data has been sent or the application wants to stop the send process. If the send data process is not completed, the caller should pass an error parameter indicating why the operation was stopped. ExgLibDisconnect is responsible for completing the operation and closing any communication ports if necessary. If data was buffered for sending in ExgLibSend, then the disconnect process may actually perform the entire transmit operation. It is important to note that the ExgLibDisconnect function can be called for an ExgLibPut, ExgLibAccept or ExgLibGet function. So it is equally important to keep track of the current operation in the ExgSocketType. If dialogs are displayed, this function must update them as appropriate. If there are errors, ExgLibDisconnect should display them (if allowed by the application).
If `ExgLibConnect` is not supported, the first call to `ExgLibPut` must clean up after itself before returning an error and the exchange library should not expect to get an `ExgLibDisconnect` call.

If `ExgLibConnect` is supported and an application calls `ExgConnect`, the exchange library should delay any cleanup until `ExgLibDisconnect`, unless `ExgLibConnect` returns an error, in which case it should clean up after itself. The library can expect to get an `ExgLibDisconnect` call if it returns `errNone` from `ExgLibConnect`. If the application does not call `ExgConnect`, the first call to `ExgLibPut` must clean up after itself before returning an error and the exchange library should not expect to get an `ExgLibDisconnect` call.

The Exchange Manager’s `ExgDisconnect` function simply calls `ExgLibDisconnect` in the exchange library identified by the `ExgSocketType` structure passed to `ExgDisconnect`.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.

---

**ExgLibGet**

**Purpose**

Establish a connection and request data from a remote device.

**Declared In**

ExgLib.h

**Prototype**

```c
Err ExgLibGet(UInt16 libRefNum, ExgSocketType *exgSocketP)
```

**Parameters**

- `libRefNum` Reference number of this exchange library.
- `exgSocketP` A pointer to the socket structure (see `ExgSocketType`).

**Result**

Returns `errNone` if no error. `exgErrNotSupported` is returned if this operation is not supported by this library. Other error codes are defined by each exchange library.

**Comments**

`ExgLibGet` informs the library that it should make a connection to the remote device and request information from it. When an
exchange library’s ExgLibGet function is called, it should fetch the requested data and prepare to deliver it when the application calls ExgReceive. After ExgLibReceive is called, possibly more than once, ExgLibDisconnect follows.

The Exchange Manager’s ExgGet function calls ExgLibGet in the exchange library identified in the ExgSocketType structure passed to ExgGet. If an exchange library’s implementation of ExgLibGet returns exgErrNotEnoughPower, the Exchange Manager puts up an alert, so there is no need for the exchange library to do so. Other error codes are not treated specially by the Exchange Manager; an exchange library must put up its own alerts when appropriate.

**Compatibility** Implemented only if 4.0 New Feature Set is present.

### ExgLibHandleEvent

**Purpose** Handle exchange-library-specific events.

**Declared In** ExgLib.h

**Prototype**

```c
Boolean ExgLibHandleEvent(UInt16 libRefnum, void *eventP)
```

**Parameters**

- `libRefnum` Reference number of this exchange library.
- `eventP` The event to handle.

**Result** Returns true if the event was handled.

**Comments** Exchange libraries are free to implement this function for internal or external use. The Exchange Manager does not call it.
ExgLibOpen

**Purpose**  Library-specific. Although this function is not called by the Exchange Manager, all shared libraries normally implement it.

**Declared In**  ExgLib.h

**Prototype**  Err ExgLibOpen (UInt16 libRefnum)

**Parameters**  -> libRefnum  Reference number of this exchange library.

**Comments**  Exchange libraries are free to implement this function for internal or external use. The ExgLibDisconnect does not call it.
ExgLibPut

**Purpose**
Signals the start of an object to be transferred to the destination device.

**Declared In**
ExgLib.h

**Prototype**
Err ExgLibPut(UInt16 libRefnum,
ExgSocketType *exgSocketP)

**Parameters**
- **libRefnum**: Reference number of this exchange library.
- **exgSocketP**: Pointer to the socket structure (see ExgSocketType).

**Result**
Returns errNone if no error. Error codes are defined by each exchange library.

**Comments**
Opens a connection if necessary. The actual data should be sent using ExgLibSend after which the connection should be shut down with ExgLibDisconnect. The exchange library may prompt the user for addressing information.

The first time this library is called, it may be necessary to allocate global variables and perform other initialization steps. It is usually a good idea to keep any state information about the open connection in the socketRef field of the exgSocketP structure. This data can then be passed to subsequent operations on that socket.

ExgLibPut should then check if the socketRef field of exgSocketP has been initialized. In some cases, an application may already have filled socketRef with addressing information. The use of socketRef is entirely up to the exchange library. If socketRef is empty, the exchange library needs to fill in any addressing information. In order to do this, the exchange library needs to open its own dialog asking the user for whatever addressing information would be appropriate.

Exchange libraries are responsible for any validation of data entered in the addressing dialog and may use Address Book lookup or other system features to improve the user experience.
Once the user has completed addressing and confirmed the dialog, the exchange library should, in general, call the Progress Manager to open a progress dialog that remains open during the entire put operation. The progress dialog should not be opened if the noStatus option was passed or if the transaction is in asynchronous mode.

If the exchange library is displaying dialogs, it must also look for events and pass them to the Progress Manager.

Other operations within ExgLibPut depend on the exchange library. The exchange library may open communications ports and establish remote links at this time. Or it may just open a stream for buffering data until a later operation. If any errors occur in this process, the exchange library is responsible for removing any progress dialogs and returning an error. If displaying progress, the exchange library may also need to display an error using the Progress Manager before returning.

**NOTE:** The progress dialog is converted to an error dialog and waits until the user dismisses it. This may occur in ExgLibPut or later in ExgLibDisconnect. It depends on whether the connection was made in ExgLibPut or whether it used an existing connection made by a previous call to ExgLibConnect or ExgLibPut.

The first call to ExgLibPut must clean up after itself before returning an error if one of the following conditions exists:

- If ExgLibConnect is not supported.
- If ExgLibConnect is supported and an application does not call ExgLibConnect.

The Exchange Manager’s ExqPut function calls ExgLibPut in the exchange library identified by the ExqSocketType structure passed to ExqPut. If ExgLibPut returns exgErrNotEnoughPower, the Exchange Manager puts up an alert, so there is no need for the exchange library to do so. Other error codes are not treated specially by the Exchange Manager; an exchange library must put up its own alerts when appropriate.
**Exchange Library**

**Exchange Library Functions**

---

**Compatibility**

Implemented only if 4.0 New Feature Set is present.

---

**ExgLibReceive**

**Purpose**

Receive data from a remote device.

**Declared In**

ExgLib.h

**Prototype**

```c
UInt32 ExgLibReceive(UInt16 libRefNum,
ExgSocketType *exgSocketP, void *bufP,
UInt32 bufSize, Err *errP)
```

**Parameters**

- `libRefNum` Reference number of this exchange library.
- `exgSocketP` A pointer to the socket structure (see ExgSocketType).
- `bufP` A pointer to a buffer into which the data is put.
- `bufSize` The size of the buffer in bytes.
- `errP` The error code result: errNone if no error. exgErrNotSupported is returned if used during a preview and the exchange library does not support preview or if there appears to be more data available but ExgLibReceive cannot obtain it. Error codes are defined by each exchange library.

**Result**

The number of bytes received. Returns 0 if the object is complete. ExgLibReceive blocks until at least one byte is available or the object is complete.

**Comments**

Use after ExgLibGet or ExgLibAccept to receive the contents of the object. May be used after ExgLibAccept to examine the contents during a preview.

ExgLibReceive must update any progress dialogs to indicate that data is being received. The ExgLibReceive should fill the buffer passed as much as possible and return the actual number of bytes that were stored in the buffer. ExgLibReceive must block for at
least one byte if the stream is not complete. Returning zero bytes indicates the end of the data.

The Exchange Manager’s `ExgReceive` function simply calls `ExgLibReceive` in the exchange library identified by the `ExgSocketType` structure passed to `ExgReceive`.

**Compatibility**

Implemented only if [4.0 New Feature Set](#) is present.

**ExgLibRequest**

**Purpose**

Requests an object using a URL, then has the Exchange Manager send it to the default application registered for the object’s type.

**Declared In**

`ExgLib.h`

**Prototype**

```c
Err ExgLibRequest(UInt16 libRefNum, ExgSocketType *socketP)
```

**Parameters**

- `libRefNum` Reference number of this exchange library.
- `socketP` Pointer to the socket structure (see `ExgSocketType`). The name field contains the URL of the object being requested.

**Result**

Returns `errNone` if no error. `exgErrNotSupported` is returned if the exchange library does not support this operation. Other error codes are defined by each exchange library.

**Comments**

The exchange library may prompt the user for addressing information. The socket’s name field may contain a URL with any of the schemes for which the exchange library registered. Not all exchange libraries support this operation.

**NOTE:** This function is often used to tickle an exchange library to make it check to see if there are new messages. These messages are then delivered as usual. So there may or may not be a specific object being requested.
The Exchange Manager’s ExgRequest function calls ExgLibRequest in the exchange library identified by the ExgSocketType structure passed to ExgRequest. If ExgLibRequest returns exgErrNotEnoughPower, the Exchange Manager puts up an alert, so there is no need for the exchange library to do so. Other error codes are not treated specially by the Exchange Manager; an exchange library must put up its own alerts when appropriate.

See Also ExgRegisterDatatype

**ExgLibSend**

**Purpose** Send data for an object to a destination device.

**Declared In** ExgLib.h

**Prototype**

```c
UInt32 ExgLibSend(UInt16 libRefNum, ExgSocketType *exgSocketP, const void *bufP, UInt32 bufLen, Err *errP)
```

**Parameters**

- `libRefNum` Reference number of this exchange library.
- `exgSocketP` A pointer to the socket structure (see ExgSocketType).
- `bufP` A pointer to a buffer containing the data to send.
- `bufLen` The number of bytes to send.
- `errP` The error code result. Error codes are defined by each exchange library

**Result** Returns the number of bytes sent.

**Comments** Applications call ExgSend after ExgPut and in response to sysAppLaunchCmdExgGetData.

This function blocks until all bytes to be sent are actually sent, or until an error occurs (such as device full).
ExgLibSend may be called any number of times with varying size buffers to transmit information. If dialogs are being displayed, ExgLibSend must keep them updated (perhaps with animation or progress information). ExgLibSend must also check for events and let the Progress Manager handle them.

The Exchange Manager’s ExgSend function simply calls ExgLibSend in the exchange library identified by the ExgSocketType structure passed to ExgSend.

**Compatibility** Implemented only if 4.0 New Feature Set is present.

**ExgLibSleep**

**Purpose** The device is going to sleep.

**Declared In** ExgLib.h

**Prototype** Err ExgLibSleep(UInt16 libRefnum)

**Parameters**

- `libRefnum` Reference number of this exchange library.

**Result** Returns errNone if no error. Error codes are defined by each exchange library.

**Comments** The device goes into sleep mode when the user turns the device off, the auto-off timer expires, or power is low. All shared libraries must implement ExgLibSleep; however, no processing is required and simply returning errNone is enough.

**IMPORTANT:** Libraries must return from this function quickly to allow sufficient time for emergency shutdown situations (removal of batteries, for example).
ExgLibWake

Purpose  The device is waking up.

Declared In  ExgLib.h

Prototype  Err ExgLibWake(UInt16 libRefnum)

Parameters  

Result  Returns errNone if no error. Error codes are defined by each exchange library.

Comments  The device wakes up when the user turns the device on. All shared libraries must implement ExgLibWake, although no processing is required; it is enough to simply return errNone.
IR Library

The IR (InfraRed) library is a shared library that provides a direct interface to the IR communications capabilities of the Palm OS®. This chapter provides reference material for the IR library API:

- IR Library Data Structures
- IR Library Constants
- IR Stack Callback Events
- IR Library Functions
- IAS Functions
- Application-Defined Functions

The header file `irlib.h` declares the IR library API. For more information on the IR library, see the chapter “Beaming (Infrared Communication)” in the Palm OS Programmer’s Companion, vol. II, Communications.

IR Library Data Structures

This section lists some of the more important data types used by the IR library functions.

**IrConnect**

The `IrConnect` structure is used to manage an IrLMP or Tiny TP connection.

```c
typedef struct _hconnect {
    UInt8 lLsap;
    UInt8 rLsap;
    UInt8 flags;
    UInt8 reserved;
    IrCallBack callBack;
    IrPacket packet;
    ListEntry packets;
} irConnect;
```
Field Descriptions

IrPacket

The IrPacket structure is used for sending IrDA packets.

typedef struct _IrPacket {
    ListEntry node;
    UInt8 *buff;
    UInt16 len;
    IrConnect* origin;
    UInt8 headerLen;
    UInt8 header[14];
    UInt8 reserved;
} IrPacket;

Field Descriptions

node For system use only.

buff Pointer to the send data buffer.
IMPORTANT: The node field must be the first field in the structure. It is used internally by the stack.

IrIASObject

The IrIASObject structure is used as storage for an IAS object managed by the local IAS server. An object of this type is passed as the obj parameter to the IrIAS_Add function.

```c
typedef struct _IrIasObject {
    UInt8 *name;
    UInt8 len;
    UInt8 nAttribs;
    IrIasAttribute* attribs;
} IrIasObject;
```

Field Descriptions

- **name**: Pointer to name of object.
- **len**: Length of object name.
- **nAttribs**: Number of attributes.
- **attribs**: Pointer to an array of attributes.
IrlasQuery

The IrlasQuery structure is used to perform IAS queries. The IrlasQuery object is passed as the token parameter to functions such as IRIAS_Query and IRIAS_Next.

typedef struct _IrlasQuery {
    UInt8 queryLen;
    UInt8 reserved;
    UInt8 *queryBuf;
    UInt16 resultBufSize;
    UInt16 resultLen;
    UInt16 listLen;
    UInt16 offset;
    UInt8 retCode;
    UInt8 overFlow;
    UInt8 *result;
    IrlasQueryCallBack callBack;
} _IrlasQuery;

Field Descriptions

queryLen Total length of the query.
reserved Reserved for future use.
queryBuf Pointer to buffer containing the query.
resultBufSize Size of the result buffer.
resultLen Actual number of bytes in the result buffer.
listLen Number of items in the result list.
offset Offset into the results buffer.
retCode Return code of operation.
overFlow Set to true if result exceeded result buffer size.
result Pointer to buffer containing result.
callBack Pointer to query callback function.
IrCallbackParms

The IrCallbackParms structure is used to pass information from the stack to the upper layer of the stack (application). Not all fields are valid at any given time. The type of event determines which fields are valid. The IrCallbackParms object is passed as the second parameter to the IrCallback function.

```c
typedef struct {
    IrEvent event;
    UInt8 reserved1;
    UInt8  *rxBuff;
    UInt16  rxLen;
    IrPacket* packet;
    IrDeviceList* deviceList;
    IrStatus status;
    UInt8 reserved2;
} IrCallbackParms;
```

Field Descriptions

- **event**: Event causing the callback.
- **reserved1**: Reserved for future use.
- **rxBuff**: Received data buffer.
- **rxLen**: Length of data in received buffer.
- **packet**: Pointer to packet being returned.
- **deviceList**: Pointer to discovery device list.
- **status**: Status of stack.
- **reserved2**: Reserved for future use.

IrStatsType

The IrStatsType structure defines performance statistics for the IR Library. Use the `ExgControl` function with an `irGetStatistics` operation to retrieve these statistics. See IR Control Constants for more information.
typedef struct {
    UInt16 recLineErrors;
    UInt16 crcErrors;
} IrStatsType;

Field Descriptions

recLineErrors The number of serial errors since the library opened.

crcErrors The number of CRC errors since the library opened.

IR Library Constants

IR Control Constants

The IR control constants define operations that the IR Exchange Library can perform. You pass these constants as the operation parameter to `ExgControl`. The following table lists the operation constants, the data that should be passed as the `valueP` parameter to `ExgControl`, and what operation is performed in response.

<table>
<thead>
<tr>
<th>Operation Constant</th>
<th>value Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>irGetScanningMode</td>
<td>Boolean. Output only.</td>
<td>Returns true in <code>*valueP</code> if beaming is enabled or false if beaming is disabled.</td>
</tr>
<tr>
<td>irGetStatistics</td>
<td><code>IrStatsType</code>. Output only.</td>
<td>Returns performance statistics.</td>
</tr>
<tr>
<td>irRestoreScanning</td>
<td>None</td>
<td>Re-enables beaming after an <code>irSuppressScanning</code> operation. This operation keeps track of the number of requests that beaming be disabled and re-enables beaming only when the count reaches 0.</td>
</tr>
</tbody>
</table>
This operation differs from `irSetScanningMode` in that it does not update the saved preferences.

This operation modifies the saved preferences database, which is back up during a HotSync® operation. Because of this, beaming may remain disabled after a reset if you use this operation to disable it. If you want to temporarily disable beaming use `irSuppressScanning` and `irRestoreScanning` instead.

Sets the possible baud rates that the IR Library will use to those specified in `*valueP`. OR the `irOpenOptSpeed...` constants together to specify more than one. The default rate is 0, which causes the baud rate to be determined by the hardware.

This operation is sometimes useful for debugging connections. Generally, you should set all bits up to the fastest rate you want to allow. To reset, use this operation again and pass 0 in `*valueP`.

If you change the baud rate, your changes are until the device is reset or you perform this operation again.
IR Stack Callback Events

The IR stack calls the application by way of a callback function stored in each `IrConnect` structure. The callback function is called with a pointer to the `IrConnect` structure and a pointer to a parameter structure. The parameter structure contains an event field, which indicates the reason the callback is called, and other parameters, which have meaning based on the event.

### Operation Constant | value | Data Type | Description
--- | --- | --- | ---
`irSetSerialMode` | Boolean | Input only | If the specified value is `true`, the IR Library uses the serial port instead of the infrared port until the device is reset. This option is useful for debugging. You can run your application in POSE and use the IR Library to communicate with a device connected in the cradle.

`irSetSupported` | Boolean | Input only | If `true`, IR is supported on this device. If `false`, IR is not supported. You can use this constant to disable the unsupported dialog that normally displays when a beam is attempted and no IR support is available.

`irSuppressScanning` | None | | Temporarily disables beam receive. This operation keeps track of the number of requests that beaming be disabled and re-enables beaming (through `irRestoreScanning`) only when the count reaches 0.

This operation differs from `irSetScanningMode` in that it does not update the saved preferences.
The meaning of the events is described in the following sections.

**LEVENT_DATA_IND**
Data has been received. The received data is accessed using fields `rxBuff` and `rxLen`.

**LEVENT_DISCOVERY_CNF**
Indicates the completion of a discovery operation. The field `deviceList` points to the discovery list.

**LEVENT_LAP_CON_CNF**
The requested IrLAP connection has been made successfully. The callback function of all bound `IrConnect` structures is called.

**LEVENT_LAP_CON_IND**
Indicates that the IrLAP connection has come up. The callback of all bound `IrConnect` structures is called.

**LEVENT_LAP_DISCON_IND**
Indicates that the IrLAP connection has gone down. This means that all IrLMP connections are also down. A callback with event `LEVENT_LM_CON_IND` is not given. The callback function of all bound `IrConnect` structures is called.

**LEVENT_LM_CON_CNF**
The requested IrLMP/Tiny TP connection has been made successfully. Connection data from the other side is found using fields `rxBuff` and `rxLen`.

**LEVENT_LM_CON_IND**
Other device has initiated a connection. `IrConnectRsp` should be called to accept the connection. Any data associated with the connection request can be found using fields `rxBuff` and `rxLen`, data pointer and length, respectively.
LEVENT_LM_DISCON_IND
The IrLMP/Tiny TP connection has been disconnected. Any data associated with the disconnect indication can be found using fields rxBuff and rxLen, data pointer and length, respectively.

LEVENT_PACKET_HANDLED
A packet is being returned. A pointer to the packet exists in field packet.

LEVENT_STATUS_IND
Indicates that a status event from the stack has occurred. The status field indicates the status generating the event. Possible status values are as follows:

- IR_STATUS_NO_PROGRESS which means that IrLAP has no progress for 3 seconds threshold time (for example, the beam is blocked).
- IR_STATUS_LINK_OK which indicates that the no progress condition has cleared.
- IR_STATUS_MEDIA_NOT_BUSY which indicates that the IR media has transitioned from busy to not busy.

LEVENT_TEST_CNF
Indicates that a TEST command has completed. The status field indicates if the test was successful.

- IR_STATUS_SUCCESS indicates that the operation was successful and the data in the test response can be found by using the rxBuff and rxLen fields.
- IR_STATUS_FAILED indicates that no TEST response was received. The packet passed to perform the test command is passed back in the packet field and is now available (no separate packet handled event occurs).

LEVENT_TEST_IND
Indicates that a TEST command frame has been received. A pointer to the received data is in rxBuff and rxLen. A pointer to the
packet that is sent in response to the test command is in the packet field. The packet is currently set up to respond with the same data sent in the command TEST frame. If different data is desired as a response, then you need to modify the packet structure. This event is sent to the callback function in all bound `IrConnect` structures. The IAS connections ignore this event.

## IR Library Functions

### IrAdvanceCredit

**Purpose**
Advances the credit to the other side of the connection.

**Declared In**
`IrLib.h`

**Prototype**
```c
void IrAdvanceCredit (IrConnect* con,UInt8 credit)
```

**Parameters**
- `con`
  Pointer to `IrConnect` structure representing connection to which credit is advanced.
- `credit`
  Amount of credit to advance.

**Result**
Returns nothing.

**Comments**
The credit passed by this function is added to the existing available credit, which must not exceed 127. This function only makes sense for a Tiny TP connection.

**Compatibility**
Implemented only if 3.0 New Feature Set is present.
**IrBind**

**Purpose**
Obtains a local LSAP selector and registers the connection with the protocol stack.

**Declared In**
IrLib.h

**Prototype**
IrStatus IrBind (UInt16 refNum, IrConnect* con, IrCallBack callBack)

**Parameters**
- `refnum` IR library refNum.
- `con` Pointer to `IrConnect` structure.
- `callBack` Pointer to a `callBack` function that handles the indications and confirmation from the protocol stack.

**Result**
- `IR_STATUS_SUCCESS` means the operation completed successfully. The assigned LSAP can be found in `con->lLsap`.
- `IR_STATUS_FAILED` means the operation failed for one of the following reasons:
  - `con` is already bound to the stack.
  - There is no room in the connection table.

**Comments**
The `IrConnect` structure is re-initialized. Any values stored in the structure are lost. The assigned LSAP is returned in the `lLsap` field of `con`. The type of the connection is set to IrLMP. The `IrConnect` must be bound to the stack before it can be used.

**Compatibility**
Implemented only if 3.0 New Feature Set is present.
**IrClose**

**Purpose** Closes the IR library. This releases the global memory for the IR stack and any system resources it uses. This must be called when an application is done with the IR library.

**Declared In** IrLib.h

**Prototype** Err IrClose (UInt16 refnum)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refnum</td>
<td>IR library refNum.</td>
</tr>
</tbody>
</table>

**Result** Returns 0 if successful.

**Comments** Do not call this function unless the call to **IrOpen** was successful.

**Compatibility** Implemented only if 3.0 New Feature Set is present.

---

**IrConnectIrLap**

**Purpose** Starts an IrLAP connection.

**Declared In** IrLib.h

**Prototype** IrStatus IrConnectIrLap (UInt16 refNum, IrDeviceAddr deviceAddr)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refnum</td>
<td>IR library refNum.</td>
</tr>
<tr>
<td>deviceAddr</td>
<td>32-bit address of device to which connection should be made.</td>
</tr>
</tbody>
</table>

**Result**

- **IR_STATUS_PENDING** means the operation started successfully; the result is returned by way of a callback.
- **IR_STATUS_MEDIA_BUSY** means the operation failed because the media is busy. Media busy is caused by one of the following reasons:
  - Other devices are using the IR medium.
• An IrLAP connection already exists.
• A discovery process is in progress.

Comments  
The result is signaled to all bound IrConnect structures by way of the callback function. The callback event is LEVENT_LAP_CON_CNF if successful or LEVENT_LAP_DISCON_IND if unsuccessful.

Compatibility  
Implemented only if 3.0 New Feature Set is present.

IrConnectReq

Purpose  
Requests an IrLMP or Tiny TP connection.

Declared In  
IrLib.h

Prototype  
IrStatus IrConnectReq (UInt16 refNum,  
IrConnect* con, IrPacket* packet, UInt8 credit)

Parameters  
---> refnum  
IR library refNum.

---> con  
Pointer to IrConnect structure for handling the connection. The rLsap field must contain the LSAP selector for the peer on the other device. Also the type of the connection must be set. Use IR_SetConTypeLMP to set the type to an IrLMP connection or IR_SetConTypeTTP to set the type to a Tiny TP connection.

---> packet  
Pointer to a packet that contains connection data. Even if no connection data is needed, the packet must point to a valid IrPacket structure. The packet is returned by way of the callback function with the LEVENT_PACKET_HANDLED event if no errors occur. The maximum size of the packet is IR_MAX_CON_PACKET for an IrLMP connection or IR_MAX_TTP_CON_PACKET for a Tiny TP connection.
IR Library
IR Library Functions

--> credit

Initial amount of credit advanced to the other side. Must be less than 127. It is ANDed with 0x7f, so if it is greater than 127, unexpected results occur. This parameter is ignored if the connection is an IrLMP connection.

Result

IR_STATUS_PENDING means the operation has been started successfully and the result is returned by way of the callback function with the event LEVENT_LM_CON_CNF if the connection is made or LEVENT_LM_DISCON_IND if connection fails. The packet is returned by way of the callback with the event LEVENT_PACKET_HANDLED.

IR_STATUS_FAILED means the operation failed because of one of the following reasons. Note that the packet is available immediately.

- The connection is busy (already involved in a connection).
- The IrConnect structure is not bound to the stack.
- The packet size exceeds maximum allowed.

IR_STATUS_NO_IRLAP means the operation failed because there is no IrLAP connection (the packet is available immediately).

Comments

The result is signaled by way of the callback specified in the IrConnect structure. The callback event LEVENT_LM_CON_CNF indicates that the connection is up and LEVENT_LM_DISCON_IND indicates that the connection failed. Before calling this function the fields in the con structure must be properly set.

Compatibility

Implemented only if 3.0 New Feature Set is present.
IrConnectRsp

Purpose   Accepts an incoming connection that has been signaled by way of the callback with the event LEVENT_LM_CON_IND.

Declared In  IrLib.h

Prototype  IrStatus IrConnectRsp (UInt16 refNum,
IrConnect* con, IrPacket* packet, UInt8 credit)

Parameters

--> refnum    IR library refNum.

--> con    Pointer to IrConnect structure.

--> packet    Pointer to a packet that contains connection data. Even if no connection data is needed, the packet must point to a valid IrPacket structure. The packet is returned by way of the callback with the LEVENT_PACKET_HANDLED event if no errors occur. The maximum size of the packet is IR_MAX_CON_PACKET for an IrLMP connection or IR_MAX_TTP_CON_PACKET for a Tiny TP connection.

--> credit    Initial amount of credit advanced to the other side. Must be less than 127. It is ANDed with 0x7f, so if it is greater than 127, unexpected results occur. This parameter is ignored if the connection is an IrLMP connection.

Result  IR_STATUS_PENDING means the operation has been started successfully and the packet is returned by way of the callback function with the event LEVENT_PACKET_HANDLED.

IR_STATUS_FAILED means the operation failed because of one of the following reasons. Note that the packet is available immediately.

• The connection is not in the proper state to require a response.

• The IrConnect structure is not bound to the stack.
• The packet size exceeds the maximum allowed.

IR_STATUS_NO_IRLAP means the operation failed because there is no IrLAP connection (the packet is available immediately).

Comments
IrConnectRsp can be called during the callback or later to accept the connection. The type of the connection must already have been set to IrLMP or Tiny TP before the LEVENT_LM_CON_IND event.

Compatibility
Implemented only if 3.0 New Feature Set is present.

IrDataReq

Purpose
Sends a data packet.

Declared In
IrLib.h

Prototype
IrStatus IrDataReq (UInt16 refNum, IrConnect* con, IrPacket* packet)

Parameters
--> refnum  IR library refNum.
--> con  Pointer to IrConnect structure that specifies the connection over which the packet should be sent.
--> packet  Pointer to a valid IrPacket structure that contains data to send. The packet should not exceed the maximum size found with IrMaxTxSize.

Result
IR_STATUS_PENDING means the packet has been queued by the stack. The packet is returned by way of the callback with event LEVENT_PACKET_HANDLED.

IR_STATUS_FAILED means the operation failed because of one of the following reasons. Note that the packet is available immediately.

• The IrConnect structure is not bound to the stack.
• The packet size exceeds the maximum allowed.
• The IrConnect structure does not represent an active connection.

Comments The packet is owned by the stack until it is returned by way of the callback with event LEVENT_PACKET_HANDLED. The largest packet that can be sent is found by calling IrMaxTxSize.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrDisconnectIrLap

Purpose Disconnects an IrLAP connection.

Declared In IrLib.h

Prototype IrStatus IrDisconnectIrLap (UInt16 refNum)

Parameters

--> refnum IR library refNum.

Result IR_STATUS_PENDING means the operation started successfully and all bound IrConnect structures are called back when complete.
IR_STATUS_NO_IRLAP means the operation failed because no IrLAP connection exists.

Comments When the IrLAP connection goes down, the callback of all bound IrConnect structures is called with event LEVENT_LAP_DISCON_IND.

Compatibility Implemented only if 3.0 New Feature Set is present.
IrDiscoverReq

**Purpose**
Starts an IrLMP discovery process.

**Declared In**
IrLib.h

**Prototype**
```c
IrStatus IrDiscoverReq (UInt16 refNum, 
IrConnect* con)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refnum</td>
<td>IR library refNum.</td>
</tr>
<tr>
<td>con</td>
<td>Pointer to a bound IrConnect structure.</td>
</tr>
</tbody>
</table>

**Result**

- **IR_STATUS_PENDING** means the operation is started successfully; the result is returned by way of callback.
- **IR_STATUS_MEDIA_BUSY** means the operation failed because the media is busy. Media busy is caused by one of the following reasons:
  - Other devices are using the IR medium.
  - A discovery process is already in progress.
  - An IrLAP connection exists.
- **IR_STATUS_FAILED** means the operation failed because the IrConnect structure is not bound to the stack.

**Comments**
The result is signaled by way of the callback function specified in the IrConnect structure with the event LEVENT_DISCOVERY_CNF. Only one discovery can be invoked at a time.

**Compatibility**
Implemented only if 3.0 New Feature Set is present.
**IrIsIrLapConnected**

**Purpose** Determines if an IrLAP connection exists.

**Declared In** IrLib.h

**Prototype**

```c
BOOL IrIsIrLapConnected (UInt16 refNum)
```

**Parameters**

- `refnum` IR library refNum.

**Result**

- `true` if IrLAP is connected,
- `false` otherwise.

**Comments** Only available if `IR_IS_LAP_FUNCS` is defined.

**Compatibility** Implemented only if 3.0 New Feature Set is present.

---

**IrIsMediaBusy**

**Purpose** Determines if the IR media is busy.

**Declared In** IrLib.h

**Prototype**

```c
BOOL IrIsMediaBusy (UInt16 refNum)
```

**Parameters**

- `refnum` IR library refNum.

**Result**

- `true` if IR media is busy,
- `false` otherwise.

**Comments**

Only available if `IR_IS_LAP_FUNCS` is defined.

**Compatibility**

Implemented only if 3.0 New Feature Set is present.
**IrIsNoProgress**

**Purpose**
Determines if IrLAP is not making progress.

**Declared In**
IrLib.h

**Prototype**
BOOL IrIsNoProgress (UInt16 refNum)

**Parameters**
-->
refnum
IR library refNum.

**Result**
true if IrLAP is not making progress, false otherwise.

**Compatibility**
Implemented only if 3.0 New Feature Set is present.

**IrIsRemoteBusy**

**Purpose**
Determines if IrLAP of the other device is busy.

**Declared In**
IrLib.h

**Prototype**
BOOL IrIsRemoteBusy (UInt16 refNum)

**Parameters**
-->
refnum
IR library refNum.

**Result**
true if IrLAP of the other device is busy, false otherwise.

**Compatibility**
Implemented only if 3.0 New Feature Set is present.

**IrLocalBusy**

**Purpose**
Sets the IrLAP local busy flag.

**Declared In**
IrLib.h

**Prototype**
void IrLocalBusy (UInt16 refNum, BOOL flag)

**Parameters**
-->
refnum
IR library refNum.
IR Library
IR Library Functions

-> flag Value (true or false) to set for local busy flag of IrLAP.

Result Returns nothing.

Comments If local busy is set to true, then the local IrLAP layer sends RNR (Receive Not Ready) frames to the other side indicating it cannot receive any more data. If the local busy is set to false, IrLAP is ready to receive frames.

The setting takes effect the next time IrLAP sends an RR (Receive Ready) frame. If IrLAP has data to send, the data is sent first, so it should be used carefully.

This function should not be used when using Tiny TP or when multiple connections exist.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrMaxRxSize

Purpose Returns the maximum size buffer that can be sent by the other device.

Declared In IrLib.h

Prototype(UInt16 IrMaxRxSize (UInt16 refNum, IrConnect* con))

Parameters --> refnum IR library refNum.
                --> con Pointer to IrConnect structure that represents an active connection.

Result Returns the maximum size buffer that can be sent by the other device (maximum bytes that can be received). The value returned is only valid for active connections. The maximum size varies for each connection and is based on the negotiated IrLAP parameters and the type of the connection.
Compatibility Implemented only if 3.0 New Feature Set is present.

**IrMaxTxSize**

**Purpose** Returns the maximum size allowed for a transmit packet.

**Declared In** IrLib.h

**Prototype** UInt16 IrMaxTxSize (UInt16 refNum, IrConnect* con)

**Parameters**

---

> --refnum IR library refNum.
> --con Pointer to IrConnect structure that represents an active connection.

**Result** Returns the maximum size allowed for a transmit packet. The value returned is only valid for active connections. The maximum size varies for each connection and is based on the negotiated IrLAP parameters and the type of the connection.

**Compatibility** Implemented only if 3.0 New Feature Set is present.

**IrOpen**

**Purpose** Opens the IR library. This allocates the global memory for the IR stack and reserves the system resources it requires. This must be done before any other IR library calls are made.

**Declared In** IrLib.h

**Prototype** Err IrOpen (UInt16 refnum, UInt32 options)

**Parameters**

---

> --refnum IR library refNum. This value is returned from the function SysLibFind, which you must call first to load the IR library.
IR Library Functions

---

IR Library

-- options

Open options flags. See the Comments section for details.

Result

Returns 0 if successful.

Comments

The following flags can be specified for the options parameter to set the speed of the connection:

- `irOpenOptSpeed115200` Set to maximum negotiated baud rate.
- `irOpenOptSpeed57600` Set to 57600 bps (default if no flags given).
- `irOpenOptSpeed9600` Set to 9600 bps.

Compatibility

Implemented only if 3.0 New Feature Set is present.

IrSetConTypeLMP

Purpose

Sets the type of the connection to IrLMP. This function must be called after the IrConnect structure is bound to the stack.

Declared In

IrLib.h

Prototype

void IrSetConTypeLMP (IrConnect* con)

Parameters

---> con

Pointer to IrConnect structure.

Result

Returns nothing.

Compatibility

Implemented only if 3.0 New Feature Set is present.
IrSetConTypeTTP

**Purpose**  Sets the type of the connection to Tiny TP. This function must be called after the IrConnect structure is bound to the stack.

**Declared In**  IrLib.h

**Prototype**  void IrSetConTypeTTP (IrConnect* con)

**Parameters**  

--&gt; con  Pointer to IrConnect structure.

**Result**  Returns nothing.

**Compatibility**  Implemented only if [3.0 New Feature Set](#) is present.

IrSetDeviceInfo

**Purpose**  Sets the XID info string used during discovery to the given string and length.

**Declared In**  IrLib.h

**Prototype**  IrStatus IrSetDeviceInfo (UInt16 refNum, UInt8 *info, UInt8 len)

**Parameters**  

--&gt; refnum  IR library refNum.

--&gt; info  Pointer to array of bytes.

--&gt; len  Number of bytes pointed to by info.

**Result**  IR_STATUS_SUCCESS means the operation is successful.  
IR_STATUS_FAILED means the operation failed because info is too big.

**Comments**  
The XID info string contains hints and the nickname of the device.  
The size cannot exceed IR_MAX_DEVICE_INFO bytes.
Compatibility Implemented only if 3.0 New Feature Set is present.

IrTestReq

Purpose Requests a TEST command frame be sent in the NDM (Normal Disconnect Mode) state.

Declared In IrLib.h

Prototype

IrStatus IrTestReq (UInt16 refNum, IrDeviceAddr devAddr, IrConnect* con, IrPacket* packet)

Parameters

--> refNum IR library refNum.

--> devAddr Address of device where TEST is sent. This address is not checked so it can be the broadcast address or 0.

--> con Pointer to IrConnect structure specifying the callback function to call to report the result.

--> packet Pointer to an IrPacket structure that contains the data to send in the TEST command packet. The maximum size data that can be sent is IR_MAX_TEST_PACKET. Even if no data is to be sent, a valid packet must be passed.

Result IR_STATUS_PENDING means the operation has been started successfully and the result is returned by way of the callback function with the event LEVENT_TEST_CNF. This is also the indication returning the packet.

IR_STATUS_FAILED means the operation failed because of one of the following reasons. Note that the packet is available immediately.

- The IrConnect structure is not bound to the stack.
- The packet size exceeds the maximum allowed.

IR_STATUS_MEDIA_BUSY means the operation failed because the media is busy or the stack is not in the NDM state (the packet is available immediately).
Comments
The result is signaled by way of the callback specified in the IrConnect structure. The callback event is LEVENT_TEST_CNF and the status field indicates the result of the operation. IR_STATUS_SUCCESS indicates success and IR_STATUS_FAILED indicates no response was received. A packet must be passed containing the data to send in the TEST frame. The packet is returned when the LEVENT_TEST_CNF event is given.

Compatibility
Implemented only if 3.0 New Feature Set is present.

IrUnbind

Purpose
Unbinds the IrConnect structure from the protocol stack, freeing its LSAP selector.

Declared In
IrLib.h

Prototype
IrStatus IrUnbind (UInt16 refNum, IrConnect* con)

Parameters
---> refnum IR library refNum.
---> con Pointer to IrConnect structure to unbind.

Result
IR_STATUS_SUCCESS means the operation completed successfully. IR_STATUS_FAILED means the operation failed for one of the following reasons:

• The IrConnect structure was not bound.
• The lLsap field contained an invalid number.

Compatibility
Implemented only if 3.0 New Feature Set is present.

IAS Functions

This section describes the following functions and macros related to IAS database:

• IrIAS_Add
• IrIAS_GetInteger
IrIAS_Add

Purpose
Adds an IAS object to the IAS Database.

Declared In
IrLib.h

Prototype
IrStatus IrIAS_Add (UInt16 refNum,
IrIasObject* obj)

Parameters
---> refnum IR library refNum.
---> obj Pointer to an IrIASObject structure.

Result
IR_STATUS_SUCCESS means the operation is successful.
IR_STATUS_FAILED means the operation failed for one of the following reasons:
• There is no space in the database.
• An entry with the same class name already exists.
• The attributes of the object violate the IrDA Lite rules (attribute name exceeds IR_MAX_IAS_NAME, or attribute value exceeds IR_MAX_IAS_ATTR_SIZE).
• The class name exceeds IR_MAX_IAS_NAME.

Comments The object is not copied, so the memory for the object must exist for as long as the object is in the database. The IAS database is designed to allow only objects with unique class names, and it checks for this. Class names and attributes names must not exceed IR_MAX_IAS_NAME. Also, attribute values must not exceed IR_MAX_IAS_ATTR_SIZE.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetInteger

Purpose Macro to return an integer value, assuming that the current result item is of type IAS_ATTRIB_INTEGER. (Call IrIAS_GetType to determine the type of the current result item.)

Declared In IrLib.h

Prototype IrIAS_GetInteger (t)

Parameters

Parameters

Result

Result Integer value returned as a UInt32.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetIntLsap

Purpose Macro to return an integer value that represents an LSAP, assuming that the current result item is of type IAS_ATTRIB_INTEGER. (Call
**IrIAS_GetType** to determine the type of the current result item.) Usually integer values returned in a query are LSAP selectors.

**Declared In**  IrLib.h

**Prototype**  IrIAS_GetIntLsap (t)

**Parameters**  --> t  Pointer to an **IrIasQuery** structure.

**Result**  Integer value returned as a Uint8.

**Compatibility**  Implemented only if 3.0 New Feature Set is present.

---

**IrIAS_GetObjectID**

**Purpose**  Macro to return the unique object ID of the current result item.

**Declared In**  IrLib.h

**Prototype**  IrIAS_GetObjectId (t)

**Parameters**  --> t  Pointer to an **IrIasQuery** structure.

**Result**  Returns the object ID as a Uint16 type.

**Compatibility**  Implemented only if 3.0 New Feature Set is present.
**IrIAS_GetOctetString**

**Purpose**  
Macro to return a pointer to an octet string, assuming that the current result item is of type IAS_ATTRIB_OCTET_STRING. (Call IrIAS_GetType to determine the type of the current result item.)

**Declared In**  
IrLib.h

**Prototype**  
IrIAS_GetOctetString (t)

**Parameters**  
--&gt; t  
Pointer to an IrIasQuery structure.

**Result**  
Pointer to octet string of type UInt8.

**Compatibility**  
Implemented only if 3.0 New Feature Set is present.

---

**IrIAS_GetOctetStringLen**

**Purpose**  
Macro to return the length of an octet string, assuming that the current result item is of type IAS_ATTRIB_OCTET_STRING. (Call IrIAS_GetType to determine the type of the current result item.)

**Declared In**  
IrLib.h

**Prototype**  
IrIAS_GetOctetStringLen (t)

**Parameters**  
--&gt; t  
Pointer to an IrIasQuery structure.

**Result**  
Length of octet string returned as a UInt16.

**Compatibility**  
Implemented only if 3.0 New Feature Set is present.
IrIAS_GetType

**Purpose**
Macro to return the type of the current result item.

**Declared In**
IrLib.h

**Prototype**
IrIAS_GetType (t)

**Parameters**
---> t Pointer to an IrIasQuery structure.

**Result**
Type of result item, such as IAS_ATTRIB_INTEGER, IAS_ATTRIB_OCTET_STRING or IAS_ATTRIB_USER_STRING. The return value is of type UInt8.

**Compatibility**
Implemented only if 3.0 New Feature Set is present.

IrIAS_GetUserString

**Purpose**
Macro to return a pointer to a user string, assuming that the current result item is of type IAS_ATTRIB_USER_STRING. (Call IrIAS_GetType to determine the type of the current result item.)

**Declared In**
IrLib.h

**Prototype**
IrIAS_GetUserString (t)

**Parameters**
---> t Pointer to an IrIasQuery structure.

**Result**
Pointer to result string of type UInt8.

**Compatibility**
Implemented only if 3.0 New Feature Set is present.

IrIAS_GetUserStringCharSet

**Purpose**
Macro to return the character set of the user string, assuming that the current result item is of type IAS_ATTRIB_USER_STRING.
(Call \texttt{IrIAS\_GetType} to determine the type of the current result item.)

\textbf{Declared In} \hspace{1cm} \texttt{IrLib.h} \\
\textbf{Prototype} \hspace{1cm} \texttt{IrIAS\_GetUserStringCharSet (t)} \\
\textbf{Parameters} \hspace{1cm} \rightarrow t \hspace{1cm} \text{Pointer to an \texttt{IrIasQuery} structure.} \\
\textbf{Result} \hspace{1cm} \text{Character set returned as an \texttt{IrCharSet} value.} \\
\textbf{Compatibility} \hspace{1cm} \text{Implemented only if 3.0 New Feature Set is present.}

\textbf{IrIAS\_GetUserStringLength}

\textbf{Purpose} \hspace{1cm} \text{Macro to return the length of a user string, assuming that the current result item is of type IAS\_ATTRIB\_USER\_STRING.} (Call \texttt{IrIAS\_GetType} to determine the type of the current result item.)

\textbf{Declared In} \hspace{1cm} \texttt{IrLib.h} \\
\textbf{Prototype} \hspace{1cm} \texttt{IrIAS\_GetUserStringLength (t)} \\
\textbf{Parameters} \hspace{1cm} \rightarrow t \hspace{1cm} \text{Pointer to an \texttt{IrIasQuery} structure.} \\
\textbf{Result} \hspace{1cm} \text{Length of user string returned as a UInt8 value.} \\
\textbf{Compatibility} \hspace{1cm} \text{Implemented only if 3.0 New Feature Set is present.}
**IrIAS_Next**

**Purpose**
Moves the internal pointer to the next result item.

**Declared In**
IrLib.h

**Prototype**
```
UInt8* IrIAS_Next (UInt16 refNum, IrIasQuery* token)
```

**Parameters**
- `refnum` IR library refNum.
- `token` Pointer to an `IrIasQuery` structure.

**Result**
Pointer to the next result item, or 0 if there are no more items.

**Comments**
This function returns a pointer to the start of the next result item. If the pointer is 0, then there are no more result items.

**Compatibility**
Implemented only if [3.0 New Feature Set](#) is present.

---

**IrIAS_Query**

**Purpose**
Makes an IAS query of the IAS database of another device.

**Declared In**
IrLib.h

**Prototype**
```
IrStatus IrIAS_Query (UInt16 refNum, IrIasQuery* token)
```

**Parameters**
- `refnum` IR library refNum.
- `token` Pointer to an `IrIasQuery` structure initialized as described in the Comments section.

**Result**
`IR_STATUS_SUCCESS` means the operation is started successfully and the result is signaled by way of the callback function.

`IR_STATUS_FAILED` means the operation failed for one of the following reasons:
• The query exceeds IR_MAX_QUERY_LEN.
• The result field of token is 0.
• The resultBufSize field of token is 0.
• The callback field of token is 0.
• A query is already in progress.

IR_STATUS_NO_IRLAP means the operation failed because there is no IrLAP connection.

Comments
An IrLAP connection must exist to the other device. The IAS query token must be initialized as described below. The result is signaled by calling the callback function whose pointer exists in the IrIasQuery structure. Only one query can be made at a time.

The IrIasQuery structure passed in the token parameter must be initialized as follows:

• Assign a pointer to a callback function in which the result is signaled.
• Set result to point to a buffer large enough to hold the result of the query.
• Set resultBufSize to the size of the result buffer.
• Set queryBuf to point to a valid query.
• Set queryLen to the number of bytes in queryBuf. The length must not exceed IR_MAX_QUERY_LEN.

Compatibility
Implemented only if 3.0 New Feature Set is present.
IrIAS_SetDeviceName

Purpose
Sets the value field of the device name attribute of the “Device” object in the IAS database.

Declared In
IrLib.h

Prototype
IrStatus IrIAS_SetDeviceName (UInt16 refNum, UInt8 *name, UInt8 len)

Parameters
--> refnum  IR library refNum.
--> name  Pointer to an IAS value field for the device name attribute of the device object. It includes the attribute type, character set and device name. This value field should be a constant and the pointer must remain valid until IrIAS_SetDeviceName is called with another pointer.
--> len  Total length of the value field. Maximum size allowed is IR_MAX_IAS_ATTR_SIZE.

Result
IR_STATUS_SUCCESS means the operation is successful.
IR_STATUS_FAILED means len is too big, or the value field is not a valid user string.

Compatibility
Implemented only if 3.0 New Feature Set is present.
IrIAS_StartResult

**Purpose**
Macro to put the internal pointer to the start of the result buffer.

**Declared In**
IrLib.h

**Prototype**
IrIAS_StartResult (t)

**Parameters**

\[ t \]
Pointer to an IrIasQuery structure.

**Result**
Returns nothing.

**Compatibility**
Implemented only if 3.0 New Feature Set is present.

Application-Defined Functions

The functions in this section are supplied by the developer and can be named anything.

IrIasQueryCallBack

**Purpose**
Signals the result of IAS query. The result of IAS queries is signaled by calling this callback function which is pointed to by the callBack field of the IrIasQuery structure.

**Declared In**
IrLib.h

**Prototype**
void IrIasQueryCallBack (IrStatus status)

**Parameters**

\[ \text{-- } \text{status} \]
The status of the query operation. The following values can be passed:

\[ \text{IR\_STATUS\_SUCCESS} \]
means the query operation finished successfully and the results can be parsed.

\[ \text{IR\_STATUS\_DISCONNECT} \]
means the link or IrLMP connection was disconnected.
IR Library
Application-Defined Functions

during the query, so the results are not valid.

Result Returns nothing.
Modem Manager

This chapter provides reference material for the modem manager API. The header file ModemMgr.h declares the modem manager API.

Modem Manager Functions

MdmDial

**Purpose**
Initialize the modem, dial the phone number and wait for result.

**Declared In**
ModemMgr.h

**Prototype**
Err MdmDial (MdmInfoPtr modemP, Char *okDialP, Char *userInitP, Char *phoneNumP)

**Parameters**
- **modemP**
  Pointer to modem info structure (filled in by caller)
- **okDialP**
  (NOT IMPLEMENTED) Pointer to string of chars allowed in dial string
- **userInitP**
  Pointer to modem setup string without the AT prefix.
- **phoneNumP**
  Pointer to phone number string

**Result**
0 if successful; otherwise mdmErrNoTone, mdmErrNoDCD, mdmErrBusy, mdmErrUserCan, mdmErrCmdError

**Comments**
When executing this function, the system performs these steps:
- Switch to the requested initial baud rate.
- If HW hand-shake is requested, enable CTS/RTS hand-shaking; otherwise, disable it.
Modem Manager
Modem Manager Functions

- Reset the modem.
- Execute the setup string (if any).
- Configure the modem with required settings.
- Dial the phone number.
- Wait for CONNECT XXXXX or other response.
- If auto-baud is requested, switch to the connected baud rate.

MdmHangUp

**Purpose**
Hang up the modem.

**Declared In**
ModemMgr.h

**Prototype**
Err MdmHangUp (MdmInfoPtr modemP)

**Parameters**
modemP Pointer to modem info structure (filled in by caller)

**Result**
0 if successful.

**WARNING!** This function alters configuration of the serial port (without restoring it).
Net Library

This chapter describes the API available in the net library and its Berkeley sockets equivalents. The header file `NetMgr.h` declares the net library API. The chapter covers:

- Net Library Data Structures
- Net Library Constants
- Net Library Functions

For more information on the net library, see the chapter “Network Communication” in the *Palm OS Programmer’s Companion*, vol. II, Communications.

**IMPORTANT:** Applications cannot directly use the net library to make wireless connections. Use the INetLib for wireless connections.

Net Library Data Structures

**New**

**NetConfigNameType**

The `NetConfigNameType` structure defines a configuration name. A configuration is a specific set of values for the net library settings. Typically, users set up configurations and assign names to them using the Network preferences panel.

```c
typedef struct {
    Char name[netConfigNameSize];
} NetConfigNameType, NetConfigNamePtr;
```

name is the configuration’s name. The `netConfigNameSize` constant is currently defined to be 32.
Compatibility
Supported only if 3.2 New Feature Set is present.

NetHostInfoBufType
The NetHostInfoBufType struct contains information about a host. The NetHostInfoType struct, which maps to the hostent struct, points to fields in this struct for its information.

typedef struct {
    NetHostInfoType hostInfo;
    Char name[netDNSMaxDomainName+1];
    Char *aliasList[netDNSMaxAliases+1];
    Char aliases[netDNSMaxAliases][netDNSMaxAliases+1];
    NetIPAddr *addressList[netDNSMaxAddresses];
    NetIPAddr address[netDNSMaxAddresses];
} NetHostInfoBufType, *NetHostInfoBufPtr;

Field Descriptions
hostInfo A NetHostInfoType struct, which maps to the Berkeley UNIX sockets hostent structure.
name Official host name.
aliasList An array of aliases for the host name.
aliases
addressList An array of pointers to 32-bit IP addresses in host byte order.
address

NetHostInfoType
The NetHostInfoType structure maps to the Berkeley UNIX sockets hostent structure. It is defined as follows:

typedef struct {
    Char *nameP;
    Char **nameAliasesP;
    UInt16 addrType;
    UInt16 addrLen;
}
```c
UInt8 **addrListP;
}

Field Descriptions

nameP Official host name.
nameAliasesP An array of aliases for the host name.
addrType The type of the return addresses. See NetSocketAddrEnum.
addrLen The length in bytes of the return addresses.
addrListP An array of pointers to addresses in host byte order.
```
**NetServInfoBufType**

The `NetServInfoBufType` struct contains information about a service. The `NetServInfoType` struct, which maps to the `servent` struct, points to fields in this struct for much of its information.

```c
struct {
   NetServInfoType servInfo;
   Char           name[netServMaxName+1];
   Char         *aliasList[netServMaxAliases+1];
   Char          aliases[netServMaxAliases][netServMaxName];
   Char          protoName[netProtoMaxName+1];
   UInt8          reserved;
} NetServInfoBufType, *NetServInfoBufPtr;
```

**Field Descriptions**

- **servInfo**
  - A `NetServInfoType` struct, which maps to the Berkeley UNIX sockets `servent` structure.

- **name**
  - Official name of the service

- **aliasList**
  - Array of aliases for the service name.

- **aliases**
  - Array of aliases for the service name.

- **protoName**
  - Name of the protocol to use.

- **reserved**
  - Reserved for future use.

**NetServInfoType**

The `NetServInfoType` structure maps to the `servent` structure in Berkeley UNIX sockets API. It contains information about a service.

```c
struct {
   Char       *nameP;
   Char       **nameAliasesP;
   UInt16     port;
   Char       *protoP;
} NetServInfoType, *NetServInfoPtr;
```
Field Descriptions

nameP  

nameAliasesP  Array of aliases for the service name.

port  

protoP  Name of the protocol to use.

NetSocketAddrEnum

The NetSocketAddrEnum enum specifies the address types supported by the net library.

typedef enum {
    netSocketAddrRaw = 0,
    netSocketAddrINET = 2
} NetSocketAddrEnum

Value Descriptions

netSocketAddrRaw  Raw address. Supported in Palm OS® version 3.0 and higher.

netSocketAddrINET  IP address.

NetSocketAddrINType

The NetSocketAddrINType struct holds an internet socket address, that is, a socket that uses one of the internet protocols. This structure directly maps to the BSD UNIX sockaddr_in structure.

typedef struct NetSocketAddrINType {
    Int16 family;
    UInt16 port;
    NetIPAddr addr;
} NetSocketAddrINType;
Field Descriptions

family  Address family in host byte order. This is either netSocketAddrINET or netSocketAddrRaw.

port  The port in network byte order.

addr  The IP address in network byte order.

NetSocketAddrRawType
The NetSocketAddrRawType structure holds a raw socket address.

typedef struct NetSocketAddrRawType {
    Int16 family;
    UInt16 ifInstance;
    UInt32 ifCreator;
} NetSocketAddrRawType;

Field Descriptions

family  Address family in host byte order. This is either netSocketAddrINET or netSocketAddrRaw.

ifInstance  The instance number of the interface that the socket uses to send and receive data.

ifCreator  The creator of the interface that the socket uses.

Compatibility  Raw sockets are supported in Palm OS version 3.0 and higher.
NetSocketAddrType
The NetSocketAddrType structure holds a generic socket address. This struct can hold any type of address including Internet addresses. It directly maps to the BSD UNIX sockaddr structure.

Note that this structure is the same size as NetSocketAddrINType and NetSocketAddrRawType. This means that one of those two structures can be used for parameters declared to be NetSocketAddrType.

typedef struct NetSocketAddrType {
    Int16 family;
    UInt8 data[14];
} NetSocketAddrType;

NetSocketRef
The NetSocketRef defines a socket descriptor. The socket descriptor is created and returned by NetLibSocketOpen. It is used in any function that requires access to a socket.

typedef Int16 NetSocketRef

NetSocketTypeEnum
The NetSocketTypeEnum enum specifies the available socket types.

typedef enum {
    netSocketTypeStream=1,
    netSocketTypeDatagram=2,
    netSocketTypeRaw=3,
    netSocketTypeReliableMsg=4
} NetSocketTypeEnum

Value Descriptions
netSocketTypeStream Streams protocol over wireline.
netSocketTypeDatagram UDP protocol.
netSocketTypeRaw Raw mode.
Net Library Constants

Configuration Aliases

A configuration is a set of specific values for the net library settings. The net library defines a set of built-in configuration aliases for common network setups. These aliases point to configurations instead of holding the actual values themselves. You can specify an alias anywhere in the API you would specify a configuration.

The constants listed here specify the alias names. Most of the net library API requires a configuration index rather than a name. Use `NetLibConfigIndexFromName` to obtain the alias’s index from the name.

- `netCfgNameDefault` The default configuration.
- `netCfgNameDefWireline` The default configuration for wireline communications.
- `netCfgNameDefWireless` The default configuration for wireless communications.
- `netCfgNameCTPWireline` The default configuration for wireline communications through the Palm Web Clipping Proxy server.
- `netCfgNameCTPWireless` The default configuration for wireless communications through the Palm Web Clipping Proxy server.

By default, `netCfgNameDefault` points to the user’s default configuration, and all other aliases point to `netCfgNameDefault` except for `netCfgNameCTPWireless`, which points to an private wireless configuration.

Compatibility

Supported on version 3.2 and later.
I/O Flags
The I/O flags specify special handling instructions to functions that send and receive data. You can OR these values together to specify more than one.

- `netIOFlagPeek`: Peek at incoming message without dequeuing it.
- `netIOFlagDontRoute`: Send without using routing. This constant is currently ignored.

Tracing Bits
The tracing bits are used to set the level of event tracing. An application can get a list of events in the trace buffer using the `NetLibMaster` call.

You can set the tracing for each network interface using `NetLibIFSettingSet` and for the net library in general with `NetLibSettingSet`.

- `netTracingErrors`: Record run-time errors. This is the default.
- `netTracingMsgs`: Record application trace messages.
- `netTracingPkts`: Record packet I/O. This bit is obsolete in versions 3.2 and higher, but is mapped to `netTracingPktIP`.
- `netTracingFuncs`: Record function flow.
- `netTracingAppMsgs`: Record application messages sent using `NetLibTracePrintf` and `NetLibTracePutS`.
- `netTracingPktIP`: Record packet I/O. If this set, the following five options are enabled.
Net Library
Net Library Functions

netTracingData40      Record the first 40 bytes of each packet sent or received. This option is mutually exclusive with netTracingData.

netTracingData        Record the entirety of each packet sent or received. This option is mutually exclusive with netTracingData40.

netTracingIFHi        Record packets sent or received at the highest layer of the network interface. This layer is just below the IP layer.

netTracingIFMid       Record packets sent or received at the layer just below the highest layer of the network interface.

netTracingIFLow       Record packets sent or received at the lowest layer of the network interface.

Compatibility
The netTracingPktXXX constants are supported only in version 3.2 devices and higher. In previous versions, specify netTracingPkts instead; only the size of the packet is recorded.

Net Library Functions

NetHToNL

Purpose      Macro that converts a 32-bit value from host to network byte order.

Declared In  NetBitUtils.h

Prototype    NetHToNL (x)

Parameters   -> x          32-bit value to convert.

Result       Returns x in network byte order.
**Sockets Equivalent**

htonl()

**See Also**  
NetNToHS, NetNToHL, NetHToNS

---

**NetHToNS**

**Purpose** Macro that converts a 16-bit value from host to network byte order.

**Declared In** NetBitUtils.h

**Prototype**  
NetHToNS (x)

**Parameters**  
-> x  
16-bit value to convert.

**Result** Returns x in network byte order.

**Sockets Equivalent**  
htons()

**See Also**  
NetNToHS, NetNToHL, NetHToNL

---

**NetLibAddrAToIN**

**Purpose** Converts an ASCII string representing a dotted decimal IP address into a 32-bit IP address in network byte order.

**Declared In** NetMgr.h

**Prototype**  
NetIPAddr NetLibAddrAToIN (UInt16 libRefnum, const Char *a)

**Parameters**  
-> libRefNum  Reference number of the net library.  
-> a  Pointer to ASCII dotted decimal string.

**Result** Returns a 32-bit network byte order IP address or -1 if a doesn’t represent a dotted decimal IP address
Net Library
Net Library Functions

Sockets Equivalent

See Also NetLibAddrINToA

NetLibAddrINToA

Purpose Converts an IP address from 32-bit network byte order into a dotted decimal ASCII string.

Declared In NetMgr.h

Prototype Char *NetLibAddrINToA (UInt16 libRefnum, NetIPAddr inet, Char *spaceP)

Parameters

  - libRefNum Reference number of the net library.
  - inet 32-bit IP address in network byte order.
  - spaceP Buffer used to hold the return value.

Result Returns in spaceP the dotted decimal ASCII string representation of the IP address.

Sockets Equivalent

See Also NetLibAddrAToIN

NetLibClose

Purpose Closes the net library.

Declared In NetMgr.h

Prototype Err NetLibClose (UInt16 libRefnum, UInt16 immediate)

Parameters

  - libRefnum Reference number of the net library.
-> immediate  If true, library will shut down immediately. If false, library will shut down only if close timer expires before another NetLibOpen is issued.

**Result**  Returns one of the following values:

- 0  Success.
- netErrNotOpen  Library was not open.
- netErrStillOpen  Not really an error; returned if library is still in use by another application.

**Sockets Equivalent**  None.

**Comments**  Applications must call this function when they no longer need the net library. If the net library open count is greater than 1 before this call is made, the count is decremented and netErrStillOpen is returned. If the open count was 1, the library takes the following action:

- If immediate is true, the library shuts down immediately. All network interfaces are brought down, the net protocol stack task is terminated, and all memory used by the net library is freed.

- If immediate is false, a close timer is created and this call returns immediately without actually bringing the net library down. Instead it leaves it up and running but marks it as in the “close-wait” state. It remains in this state until either the timer expires or another NetLibOpen is issued. If the timer expires, the library is shut down. If another NetLibOpen call is issued before the timer expires (possibly by another application), the timer is cancelled and the library is marked as fully open.

In most cases, you should pass false for immediate. This allows the user to quit one Internet application and launch another within
a short period of time without having to wait through the process of closing down and then re-establishing dial-up network connections.

See Also  
[NetLibOpen](#), [NetLibOpenCount](#)

---

**New**  
NetLibConfigAliasGet

**Purpose**  
Return the configuration that an alias points to.

**Prototype**  
Err NetLibConfigAliasGet (UInt16 refNum,  
(UInt16 aliasIndex, UInt16 *indexP,  
Boolean *isAnotherAliasP)

**Parameters**  
-> refNum  
Reference number of the net library.

-> aliasIndex  
Index of the alias.

<- indexP  
Index of the configuration pointed to by the alias.

<- isAnotherAliasP  
true if indexP is the index of another alias;  
false if indexP specifies an actual configuration.

**Result**  
Returns one of the following values:  
0  
Success.

netErrConfigNotAlias  
The configuration at aliasIndex is not an alias.

netErrOutOfCmdBlocks  
netErrParamErr  
The specified index is out of range or there is no configuration at the index.

**Sockets Equivalent**  
None
Comments  Use this routine to find out which configuration a built-in alias points to. See “Configuration Aliases” for a description of the built-in aliases.

Compatibility  Implemented only if 3.2 New Feature Set is present.

See Also  NetLibConfigAliasSet

NetLibConfigAliasSet

Purpose  Set a built-in alias to point to a defined configuration.

Prototype  Err NetLibConfigAliasSet (UInt16 refNum, UInt16 configIndex, UInt16 aliasToIndex)

Parameters  -> refNum  Reference number of the net library.
  -> configIndex  Index of the built-in alias to be set.
  -> aliasToIndex  Index of the configuration to which the alias should point. You cannot set an alias to point to itself.

Result  Returns one of the following values:

  0  Success.
  netErrConfigCantPointToAlias  The configuration at aliasToIndex is an alias that points to an alias.
  netErrConfigNotAlias  The configuration at configIndex isn’t an alias.
  netErrOutOfRangeCmdBlocks  The specified index is out of range or there’s no configuration at the index.
Sockets Equivalent

None

Comments

This function is used by the Network preferences panel when the user edits a configuration. Your application can use it to associate any of the built-in aliases with a defined configuration.

The built-in aliases are typically set up as shown in Table 61.1. In this example, applications that specify a configuration index of 0 through 3 use a configuration that the user defines. Applications that use index 4 use a private configuration created by the network library.

Table 61.1 Example Configuration Table

<table>
<thead>
<tr>
<th>Index</th>
<th>Name</th>
<th>Alias To</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>.Default</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>.DefWireline</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>.DefWireless</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>.CTPWireline</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>.CTPWireless</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>_RAMCTP</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>user-defined</td>
<td></td>
</tr>
</tbody>
</table>

An alias can point to another alias so long as the nesting level is only one deep. That is, if you point an alias to an alias, you’ll receive an error if that alias in turn points to another alias. This eliminates the possibility that an alias never resolves to an actual configuration.

Compatibility

Implemented only if 3.2 New Feature Set is present.

See Also

NetLibConfigAliasGet
NetLibConfigDelete

Purpose
Delete a configuration from the net library’s configuration table.

Prototype
Err NetLibConfigDelete (UInt16 refNum, UInt16 index)

Parameters
-> refNum Reference number of the net library.
-> index Index of the configuration to delete. You cannot delete one of the built-in aliases described in “Configuration Aliases.”

Result
Returns one of the following values:
0 Success.
netErrConfigCantDelete The configuration at index is a built-in alias.
netErrOutOfRangeCmdBlocks
netErrParamErr The specified index is out of range.

Sockets Equivalent
None

Compatibility
Implemented only if 3.2 New Feature Set Set is present.

See Also
NetLibConfigSaveAs
**Net Library**

**Net Library Functions**

---

**New**

**NetLibConfigIndexFromName**

**Purpose**
Obtain a configuration’s index given its name.

**Prototype**
```
Err NetLibConfigIndexFromName (UInt16 refNum,
NetConfigNamePtr nameP, UInt16 *indexP)
```

**Parameters**
- `refNum`  Reference number of the net library.
- `nameP`  Pointer to a configuration name. See [NetConfigNameType](#).
- `indexP`  The index of the configuration with the name `*nameP`.

**Result**
Returns one of the following values:
- `0`  Success.
- `netErrConfigNotFound`  A configuration with the specified name could not be found.
- `netErrOutOfCmdBlocks`  None

**Comments**
This function returns the index of a configuration given its name. Your application should store the configuration’s index rather than its name because a configuration’s name can change.

If you pass the name of a built-in alias in `nameP`, this function returns the index of the alias’s entry in the configuration table; it does not return the index that the alias points to. For example, if the alias `netCfgNameCTPWireless` is stored at index 4 and points to index 5, `NetLibConfigIndexFromName` returns 4. If you want to obtain the index that an alias points to, use [NetLibConfigAliasGet](#).
Compatibility  Implemented only if 3.2 New Feature Set is present.

See Also  NetLibConfigList

ISSUE  New

NetLibConfigList

Purpose  Return a list of net library configuration names.

Prototype  Err NetLibConfigList (UInt16 refNum, NetConfigNameType nameArray[], UInt16 *arrayEntriesP)

Parameters  

- refNum  Reference number of the net library.

- nameArray  The list of defined configurations. See NetConfigNameType.

- arrayEntriesP  On entry, contains the number of elements in nameArray. On return, contains the number of elements in nameArray that were actually used. The Net Library currently returns up to 16 entries. If the array is not large enough to hold all the configuration names, this function returns only as many names as the array can hold.

Result  Returns one of the following values:

- 0  Success.

- netErrOutOfCmdBlocks

Sockets Equivalent  None

Comments  Use this function to obtain a list of the names of defined network configurations and configuration aliases.
Users create specific configurations using the Network preferences panel and associate names with each configuration. This function returns the list of defined configurations.

In addition to user-defined configurations, this function also returns built-in configuration aliases and private configurations. The built-in configuration aliases are described in “Configuration Aliases.” Their actual names begin with a period (.). Private configurations have names that begin with an underscore (_).

**IMPORTANT:** If you present the list returned by this function to your application’s users, you must first filter out names beginning with a period or an underscore. These names are for internal use only.

Your application should refer to a configuration by its index rather than its name because the name can be changed. To obtain the configuration’s index from its name, use NetLibConfigIndexFromName.

**Compatibility**

Implemented only if [3.2 New Feature Set](#) is present.

**New**

NetLibConfigMakeActive

**Purpose**

Makes the specified configuration current without opening the net library.

**Prototype**

Err NetLibConfigMakeActive (UInt16 refNum, UInt16 configIndex)

**Parameters**

- `refNum` Reference number of the net library.
- `configIndex` Index of the configuration to use. An index of 0 refers to the default configuration as defined by the Network preferences panel.

**Result**

Returns one of the following values:
0 Success.
netErrBufTooSmall
netErrConfigAliasErr
netErrConfigCantDelete
netErrConfigEmpty
netErrConfigNotFound
netErrOutOfCmdBlocks
netErrParamErr
netErrPrefNotFound

**Sockets Equivalent**  None

**Comments**  This function is used mainly by the Network preferences panel when the user edits and saves network configurations. The Network preferences panel uses this function to make current the configuration the user wants to edit, set the settings appropriately, and then save the configuration using `NetLibConfigSaveAs`.

Use this routine to make a specific configuration the current configuration without opening the net library. You should not use it if the net library is already open.

Unlike `NetLibOpenConfig`, this routine does not save the current net library configuration so that it can be restored upon close.

**Compatibility**  Implemented only if 3.2 New Feature Set is present.
**New**

NetLibConfigRename

**Purpose**
Rename the specified configuration.

**Prototype**
```c
Err NetLibConfigRename (UInt16 refNum, 
UInt16 index, NetConfigNamePtr newNameP)
```

**Parameters**
- `refNum` Reference number of the net library.
- `configIndex` Index of the configuration to be renamed.
- `newNameP` Pointer to the new name. See `NetConfigNameType`. The new name must not start with a period (.) or an underscore (_).

**Result**
Returns one of the following values:
- **0** Success.
- `netErrConfigBadName` The new name begins with a period.
- `netErrConfigCantDelete` The configuration at the specified index is a built-in alias or private configuration that cannot be renamed.
- `netErrOutOfCmdBlocks` The specified index is out of range or there is no configuration at the index.

**Sockets Equivalent**
None

**Comments**
You cannot specify a name beginning with a period (.) or an underscore (_). Names beginning with a period are reserved for the built-in configuration aliases. Names beginning with an underscore are hidden configurations used internally by net library.

**Compatibility**
Implemented only if 3.2 New Feature Set is present.
New

NetLibConfigSaveAs

Purpose
Save the current net library settings as a configuration with the specified name.

Prototype
Err NetLibConfigSaveAs (UInt16 refNum, NetConfigNamePtr nameP)

Parameters
-> refNum    Reference number of the net library.
-> nameP     Pointer to a name for the configuration. See NetConfigNameType. The name must not start with a period (.) or an underscore (_).

Result
Returns one of the following values:
0            Success.
netErrConfigBadName
The specified name begins with a period or underscore.
netErrConfigTooMany
Not enough space to add another configuration. The Net Library can hold up to 16 configuration.
netErrOutOfCmdBlocks

Sockets Equivalent
None

Comments
If the name you specify already exists, its configuration is replaced with this configuration.

You cannot specify a name beginning with a period (.) or an underscore (_). Names beginning with a period are reserved for the built-in configuration aliases. Names beginning with an underscore are hidden configurations used internally by net library.

The net library assigns an index to this new configuration. The configuration's index remains constant, while its name may change.
Use NetLibConfigIndexFromName to obtain the configuration’s index.

**Compatibility**
Implemented only if 3.2 New Feature Set is present.

**See Also**
NetLibConfigDelete, NetLibConfigRename

### NetLibConnectionRefresh

**Purpose**
This routine is a convenience call for applications. It checks the status of all connections and optionally tries to open any that were closed.

**Declared In**
NetMgr.h

**Prototype**
```
Err NetLibConnectionRefresh (UInt16 refNum, Boolean refresh, UInt8 *allInterfacesUpP, UInt16 *netIFErrP)
```

**Parameters**
- `-> refnum` Reference number of the net library.
- `-> refresh` If true, any connections that aren’t currently open are opened.
- `<- allInterfacesUpP` Set to true if all connections are open.
- `<- netIFErrP` First error encountered when reopening connections that were closed. (See NetLibIFUp for a list of possible values.)

**Result**
Returns one of the following values:

- 0 Success.
- netErrBufTooSmall
- netErrOutOfCmdBlocks
- netErrNoInterfaces

**Sockets Equivalent**
None.
Comments
This function determines whether a connection is up based on the internal status of the TCP/IP stack. To test the presence of a “physical connection” (phone line, modem, serial cable), a command should be sent once it’s been determined that the logical connection is up. If the physical connection is broken, nothing returns and a timeout error eventually occurs.

NetLibDmReceive

Purpose
Receive data from a socket directly into a database record.

Declared In
NetMgr.h

Prototype
Int16 NetLibDmReceive (UInt16 libRefNum, NetSocketRef socket, void *recordP, UInt32 recordOffset, UInt16 rcvLen, UInt16 flags, void *fromAddrP, UInt16 *fromLenP, Int32 timeout, Err *errP)

Parameters
- `libRefNum` Reference number of the net library.
- `socket` Descriptor for the open socket.
- `recordP` Pointer to beginning of record to receive data into. Must be locked for use.
- `recordOffset` Offset from beginning of record to read data into.
- `rcvLen` Maximum number of bytes to read.
- `flags` One or more `netIOFlagxxx` flags. See “I/O Flags.”
- `fromAddrP` Pointer to buffer to hold address of sender (a `NetSocketAddrType` struct). Pass NULL if you don’t need sender information.
- `fromLenP` On entry, size of `fromAddrP` buffer. On exit, actual size of returned address in `fromAddrP`. Pass NULL if you don’t need sender information.
Net Library
Net Library Functions

- timeout Maximum timeout in system ticks; -1 means wait forever.
- errP Contains an error code if the return value is -1.

Result Returns the number of bytes successfully received. If the return value is 0, the socket has been shut down by the remote host. If the return value is -1, an error has occurred and errP contains one of the following values:

0 No error.
netErrTimeout Call timed out.
netErrNotOpen The referenced net library has not been opened yet.
netErrParamErr
netErrSocketNotOpen
netErrWouldBlock
netErrUserCancel
netErrOutOfMemory

Comments This call behaves similarly to NetLibReceive but reads the data directly into a database record, which is normally write-protected. The caller must pass a pointer to the start of the record and an offset into the record of where to start the read.

NetLibFinishCloseWait

Purpose Forces the net library to do a complete close if it’s currently in the close-wait state.

Declared In NetMgr.h

Prototype Err NetLibFinishCloseWait (UInt16 libRefnum)

Parameters -> libRefnum Reference number of the net library.

Result Returns one of the following values:
Net Library
Net Library Functions

Sockets Equivalent
None.

Comments
This call checks the current open state of the net library. If it’s in the close-wait state (see NetLibClose), it forces the library to perform an immediate, complete close operation.

NetLibGetHostByAddr

Purpose
Looks up a host name given its IP address.

Declared In
NetMgr.h

Prototype
NetHostInfoPtr NetLibGetHostByAddr
(UInt16 libRefNum, UInt8 *addrP, UInt16 len,
UInt16 type, NetHostInfoBufPtr bufP,
Int32 timeout, Err *errP)

Parameters
- libRefNum Reference number of the net library.
- addrP IP address of host to lookup.
- len Length, in bytes, of *addrP.
- type Type of addrP. See NetSocketAddrEnum.
- bufP Pointer to a NetHostInfoBufType struct in which to store the results of the lookup.
- timeout Maximum timeout in system ticks; -1 means wait forever.
- errP Contains an error code if the return value is 0.

Result
Returns a pointer to the NetHostInfoType portion of bufP that contains results of the lookup. If the return value is 0, an error has occurred, and errP contains one of the following values:

0 No error
netErrTimeout    Call timed out.
netErrNotOpen    The referenced net library has not been opened yet.
netErrDNSNameTooLong
netErrDNSBadName
netErrDNSLabelTooLong
netErrDNSServerFailure
netErrDNSAllocationFailure
netErrDNSTimeout
netErrDNSUnreachable
netErrDNSFormat
netErrDNSServerFailure
netErrDNSNonexistantName
netErrDNSNIY
netErrDNSRefused
netErrDNSImpossible
netErrDNSSoRRS
netErrDNSAborted
netErrDNSBadProtocol
netErrDNSTruncated
netErrDNSNoRecursion
netErrDNSIrrelevant
netErrDNSNotInLocalCache
netErrDNSNoPort

**Sockets Equivalent**

```c
struct hostent *gethostbyaddr (char *addr, int len, int type);
```

**Comments**

This call queries the domain name server(s) to look up a host name given its IP address.

**See Also**

[NetLibGetHostByName](#)
**NetLibGetHostByName**

**Purpose**
Looks up a host IP address given a host name.

**Declared In**
NetMgr.h

**Prototype**
```c
NetHostInfoPtr NetLibGetHostByName
(UInt16 libRefNum, const Char *nameP,
 NetHostInfoBufPtr bufP, Int32 timeout, Err *errP)
```

**Parameters**
- `libRefNum`:
  Reference number of the net library.
- `nameP`:
  Name of host to look up.
- `bufP`:
  Pointer to a `NetHostInfoBufType` struct in which to store the results of the lookup.
- `timeout`:
  Maximum timeout in system ticks; -1 means wait forever.
- `errP`:
  Contains an error code if the return value is 0.

**Result**
Returns a pointer to the `NetHostInfoType` portion of `bufP`, which contains results of the lookup. If the return value is 0, an error has occurred and `errP` contains one of the following values:

- 0: No error
- `netErrTimeout`: Call timed out.
- `netErrNotOpen`: The referenced net library has not been opened yet.
- `netErrDNSNameTooLong`
- `netErrDNSBadName`
- `netErrDNSLabelTooLong`
- `netErrDNSAllocationFailure`
- `netErrDNSTimeout`
- `netErrDNSUnreachable`
- `netErrDNSFormat`
- `netErrDNSServerFailure`
Equivalent

struct hostent *gethostbyname (char *name);

Comments
This call first checks the local name -> IP address host table in the
net library preferences. If the entry is not found, it then queries the
domain name server(s).

See Also
NetLibGetHostByName, NetLibGetMailExchangeByName

NetLibGetMailExchangeByName

Purpose
Looks up the name of a host to use for a given mail exchange.

Declared In
NetMgr.h

Prototype
Int16 NetLibGetMailExchangeByName
(UInt16 libRefNum, Char *mailNameP,
UInt16 maxEntries, Char hostNames[] [255+1],
UInt16 priorities[], Int32 timeout, Err *errP)

Parameters
- > libRefNum Reference number of the net library.
-> mailNameP  Name of the mail exchange to look up.
-> maxEntries  Maximum number of host names to return.
<- hostNames  Array of character strings of length 255+1. The host name results are stored in this array. This array must be able to hold at least maxEntries host names.
<- priorities  Array of words. The priorities of each host name found are stored in this array. This array must be at least maxEntries in length.
-> timeout  Maximum timeout in system ticks; -1 means wait forever.
<- errP  Contains an error code if the return value is less than 0.

Result  Returns the number of entries successfully found. If the return value is a negative number, an error has occurred, and errP contains one of the following values:

0    No error
netErrTimeout  Call timed out.
netErrNotOpen  The referenced net library has not been opened yet.
netErrDNSNameTooLong
netErrDNSBadName
netErrDNSLabelTooLong
netErrDNSAllocationFailure
netErrDNSTimeout
netErrDNSUnreachable
netErrDNSFormat
netErrDNSServerFailure
netErrDNSNonexistantName
netErrDNSNIY
netErrDNSRefused
Net Library
Net Library Functions

netErrDNSImpossible
netErrDNSNoRRS
netErrDNSAborted
netErrDNSBadProtocol
netErrDNSTruncated
netErrDNSNoRecursion
netErrDNSIrrelevant
netErrDNSNotInLocalCache
netErrDNSNoPort

Sockets Equivalent
None

Comments
This call looks up the name(s) of host(s) to use for sending an e-mail. The caller passes the name of the mail exchange in mailNameP and gets back a list of host names to which the mail message can be sent.

See Also
NetLibGetHostByAddr, NetLibGetHostByName

NetLibGetServByName

Purpose
Looks up the port number for a standard TCP/IP service, given the desired protocol.

Declared In
NetMgr.h

Prototype
NetServInfoPtr NetLibGetServByName
(UIInt16 libRefNum, const Char *servNameP, const Char *protoNameP, NetServInfoBufPtr bufP, Int32 timeout, Err *errP)

Parameters
-> libRefNum Reference number of the net library.
servNameP Name of the service to look up. Possible services are "echo", "discard", "daytime", "qotd", "chargen", "ftp-data", "ftp", "telnet", "smtp", "time", "name", "finger", "pop2", "pop3", "nntp", "imap2".

protoNameP Desired protocol to use, either "udp" or "tcp".

bufP Pointer to a NetServInfoBufType struct in which to store the results of the lookup.

timeout Maximum timeout in system ticks; -1 means wait forever.

errP Contains an error code if the return value is 0.

Result Returns a pointer to the NetServInfoType portion of bufP that contains results of the lookup. If the return value is 0, and error has occurred and errP contains one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>netErrTimeout</td>
<td>Call timed out.</td>
</tr>
<tr>
<td>netErrNotOpen</td>
<td>The referenced net library has not been opened yet.</td>
</tr>
</tbody>
</table>

Sockets Equivalent

```
struct servent *getservbyname (char *addr, char *proto);
```

Comments This call is a convenience call for looking up a standard port number given the name of a service and the protocol to use.

See Also NetLibGetHostByName
**NetLibIFAttach**

**Purpose**
Attach a new network interface.

**Declared In**
NetMgr.h

**Prototype**
Err NetLibIFAttach (UInt16 libRefNum, UInt32 ifCreator, UInt16 ifInstance, Int32 timeout)

**Parameters**
- `libRefNum` Reference number of the net library.
- `ifCreator` Creator of interface to attach.
- `ifInstance` Instance number of interface to attach. The instance number is one of the values returned by `NetLibIFGet`.
- `timeout` Timeout in ticks; -1 means infinite timeout.

**Result**
Returns one of the following values:

- 0 Success.
- `netErrInterfaceNotFound`
- `netErrTooManyInterfaces`

**Sockets Equivalent**
None

**Comments**
This call can be used to attach a new network interface to the net library. Network interfaces are self-contained databases of type 'neti'. The `ifCreator` parameter to this function is used to locate the network interface database of the given creator.

If the net library is already open when this call is made, the network interface’s database will be located and then called to initialize itself and attach itself to the protocol stack in real time. If the net library is not open when this call is made, the creator and instance number of the interface are stored in the active configuration. You need to save the active configuration using `NetLibConfigSaveAs` if you want...
the interface to be initialized and attached to the stack the next time the net library is opened.

See Also  NetLibIFGet, NetLibIFDetach

### NetLibIFDetach

**Purpose**
Detach a network interface from the protocol stack.

**Declared In**
NetMgr.h

**Prototype**
```
Err NetLibIFDetach (UInt16 libRefNum, 
                  UInt32 ifCreator, UInt16 ifInstance, 
                  Int32 timeout)
```

**Parameters**
- **libRefNum** Reference number of the net library.
- **ifCreator** Creator of interface to detach.
- **ifInstance** Instance number of interface to detach.
- **timeout** Timeout in ticks; -1 means infinite timeout.

**Result**
Returns one of the following values:

- 0 Success.
- netErrInterfaceNotFound

**Sockets Equivalent**
None

**Comments**
If the net library is already open when this call is made, the interface is brought down and detached from the protocol stack in real time. If the net library is not open when this call is made, the creator and instance number of the interface are removed from the active configuration. You need to save the active configuration using NetLibConfigSaveAs if you don’t want the interface to be attached the next time the library is opened.

See Also  NetLibIFGet, NetLibIFAttach
NetLibIFDown

**Purpose**
Bring an interface down and hang up a connection.

**Declared In**
NetMgr.h

**Prototype**
```c
Err NetLibIFDown (UInt16 libRefNum,
                 UInt32 ifCreator, UInt16 ifInstance,
                 Int32 timeout)
```

**Parameters**
- `- libRefNum` Reference number of the net library.
- `- ifCreator` Creator of interface to attach.
- `- ifInstance` Instance number of interface to attach.
- `- timeout` Timeout in ticks; -1 means wait forever.

**Result**
Returns one of the following values:
- `0` Success.
- `netErrNotOpen` The referenced net library has not been opened yet.
- `netErrInterfaceNotFound`

**Sockets Equivalent**
None

**Comments**
The net library must be open before this call can be made. For dial-up interfaces, this call terminates a connection and hangs up the modem if necessary.

`NetLibClose` automatically brings down any attached interfaces, so this routine doesn’t normally have to be called.

If the interface is already down, this routine returns immediately with no error.

**See Also**
- `NetLibIFGet`, `NetLibIFAttach`, `NetLibIFDetach`, `NetLibIFUp`
**NetLibIFGet**

**Purpose**
Get the creator and instance number of an installed interface by index.

**Declared In**
NetMgr.h

**Prototype**
```c
Err NetLibIFGet (UInt16 libRefNum, UInt16 index,
                 UInt32 *ifCreatorP, UInt16 *ifInstanceP)
```

**Parameters**
- `libRefNum` Reference number of the net library.
- `index` Index of the interface to get. Indices start at 0.
- `ifCreatorP` The interface’s creator.
- `ifInstanceP` The interface’s instance number.

**Result**
Returns one of the following values:

- 0 Success.
- netErrInvalidInterface Index too high
- netErrPrefNotFound No current value for setting.

**Sockets Equivalent**
None

**Comments**
To get a list of all installed interfaces, call this function with successively increasing indices starting from 0 until the error netErrInvalidInterface is returned.

The `ifCreator` and `ifInstance` values returned from this call can then be used with the `NetLibSettingGet` call to get more information about that particular interface.

**See Also**
**Net Lib IFSettingGet**

**Purpose**
Retrieves a network interface specific setting.

**Declared In**
NetMgr.h

**Prototype**
```c
Err NetLibIFSettingGet (UInt16 libRefNum,
                        UInt32 ifCreator, UInt16 ifInstance,
                        UInt16 setting, void *valueP, UInt16 *valueLenP)
```

**Parameters**
- `libRefNum` Reference number of the net library.
- `ifCreator` Creator of the network interface.
- `ifInstance` Instance number of the network interface.
- `setting` Setting to retrieve; one of the `NetIFSettingEnum` constants.
- `valueP` Space for return value of setting.
- `valueLenP` On entry, size of `valueP`. On exit, actual size of setting.

**Result**
Returns one of the following values:
- 0 Success.
- `netErrUnknownSetting` Invalid setting constant.
- `netErrPrefNotFound` No current value for setting.
- `netErrBufTooSmall` `valueP` was too small to hold entire setting. Setting value was truncated to fit in `valueP`.
- `netErrUnimplemented`
- `netErrInterfaceNotFound`
- `netErrBufWrongSize`

**Sockets Equivalent**
None
This call can be used to retrieve the current value of any network interface setting. The caller must pass a pointer to a buffer to hold the return value (valueP), the size of the buffer (*valueLenP), and the setting ID (setting). The setting ID is one of the constants in the NetIFSettingEnum type.

Some settings, such as the login script, are variable size. For these types of settings, you can obtain the actual size required for the buffer by passing 0 for *valueLenP. The required size is returned in valueLenP.

Table 61.2 lists the network interface settings and the size of each setting. Some are only applicable to certain types of interfaces. Settings not applicable to a specific interface can be safely ignored and not set to any particular value.
### Table 61.2 Network Interface Settings

<table>
<thead>
<tr>
<th>netIFSetting...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResetAll</td>
<td>void</td>
<td>Use with <code>NetLibIFSettingSet</code> only. This clears all other settings for the interface to their default values.</td>
</tr>
<tr>
<td>Up</td>
<td>UInt8</td>
<td>Read-only. <code>true</code> if interface is currently up.</td>
</tr>
<tr>
<td>Name</td>
<td>Char[32]</td>
<td>Read-only. Name of this interface.</td>
</tr>
<tr>
<td>ReqIPAddr</td>
<td>UInt32</td>
<td>IP address of interface.</td>
</tr>
<tr>
<td>SubnetMask</td>
<td>UInt32</td>
<td>Subnet mask for interface. Doesn’t need to be specified for PPP or SLIP type connections.</td>
</tr>
<tr>
<td>Broadcast</td>
<td>UInt32</td>
<td>Broadcast address for interface. Doesn’t need to be specified for PPP or SLIP type connections.</td>
</tr>
<tr>
<td>Username</td>
<td>Char[32]</td>
<td>User name. Only required if the login script uses the user name substitution escape sequence in it. Call <code>NetLibIFSettingSet</code> with a <code>valueLen</code> of 0 to remove this setting.</td>
</tr>
<tr>
<td>Password</td>
<td>Char[32]</td>
<td>Password. Only required if the login script uses the password substitution escape sequence in it. Call <code>NetLibIFSettingSet</code> with a <code>valueLen</code> of 0 to remove this setting. If the login script uses password substitution and no password setting is set, the user will be prompted for a password at connect time.</td>
</tr>
<tr>
<td>AuthUsername</td>
<td>Char[32]</td>
<td>Authentication user name. Only required if the authentication protocol uses a different user name than the what’s in the <code>netIFSettingUsername</code> setting. If this setting is empty (<code>valueLen</code> of 0), the Username setting will be used instead. Call <code>NetLibIFSettingSet</code> with a <code>valueLen</code> of 0 to remove this setting.</td>
</tr>
</tbody>
</table>
Table 61.2 Network Interface Settings *(continued)*

<table>
<thead>
<tr>
<th>netIFSetting...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthPassword</td>
<td>Char[32]</td>
<td>Authentication password. If “$” then the user will be prompted for the authentication password at connect time. Else, if 0 length, then the netIFSettingPassword setting or the result of its prompt will be used instead. Call NetLibIFSettingSet with a valueLen of 0 to remove this setting.</td>
</tr>
<tr>
<td>ServiceName</td>
<td>Char[]</td>
<td>Service name. Used for display purposes while showing the connection progress dialog box. Call NetLibIFSettingSet with a valueLen of 0 to remove this setting.</td>
</tr>
<tr>
<td>LoginScript</td>
<td>Char[]</td>
<td>Login script. Only required if the particular service requires a login sequence. Call NetLibIFSettingSet with a valueLen of 0 to remove this setting. See below for a description of the login script format.</td>
</tr>
<tr>
<td>ConnectLog</td>
<td>Char[]</td>
<td>Connect log. Generally, this setting is just retrieved, not set. It contains a log of events from the most recent login. To clear this setting, call NetLibIFSettingSet with a valueLen of 0.</td>
</tr>
<tr>
<td>InactivityTimeout</td>
<td>UInt16</td>
<td>Maximum number of seconds of inactivity allowed. Set to 0 to ignore.</td>
</tr>
<tr>
<td>EstablishmentTimeout</td>
<td>UInt16</td>
<td>Maximum delay, in seconds, allowed between each stage of connection establishment or login script line. Must be non-zero.</td>
</tr>
<tr>
<td>DynamicIP</td>
<td>UInt8</td>
<td>If non-zero, negotiate for an IP address. If zero, the IP address specified in the netIFSettingReqIPAddress setting will be used. Default is false.</td>
</tr>
<tr>
<td>VJCompEnable</td>
<td>UInt8</td>
<td>If non-zero, enable VJ header compression. Default is true for PPP, false for SLIP, and true for CSLIP.</td>
</tr>
</tbody>
</table>
### Table 61.2 Network Interface Settings (continued)

<table>
<thead>
<tr>
<th>netIFSetting...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VJCompSlots</td>
<td>UInt8</td>
<td>Number of slots to use for VJ compression. Default is 4 for PPP and 16 for SLIP and CSLIP. More slots require more memory so it is best to keep this number to a minimum.</td>
</tr>
<tr>
<td>MTU</td>
<td>UInt16</td>
<td>Maximum transmission unit in octets. Currently not implemented in SLIP or PPP.</td>
</tr>
<tr>
<td>AsyncCtlMap</td>
<td>UInt32</td>
<td>Bit mask of characters to escape for PPP. Default is 0.</td>
</tr>
<tr>
<td>PortNum</td>
<td>UInt16</td>
<td>Which serial communication port to use. Port 0 is the only port available on the device.</td>
</tr>
<tr>
<td>BaudRate</td>
<td>UInt32</td>
<td>Serial port baud rate to use in bits per second.</td>
</tr>
<tr>
<td>FlowControl</td>
<td>UInt8</td>
<td>If bit 0 is 1, use hardware handshaking on the serial port. Default is no hardware handshaking.</td>
</tr>
<tr>
<td>StopBits</td>
<td>UInt8</td>
<td>Number of stop bits. Default is 1.</td>
</tr>
<tr>
<td>ParityOn</td>
<td>UInt8</td>
<td>true if parity detection enabled. Default is false.</td>
</tr>
<tr>
<td>ParityEven</td>
<td>UInt8</td>
<td>true for even parity detection. Default is true.</td>
</tr>
<tr>
<td>UseModem</td>
<td>UInt8</td>
<td>If true, dial-up through modem. If false, go direct over serial port</td>
</tr>
<tr>
<td>PulseDial</td>
<td>UInt8</td>
<td>If true, pulse dial modem. Else, tone dial. Default is tone dial.</td>
</tr>
<tr>
<td>ModemInit</td>
<td>Char[]</td>
<td>Zero-terminated modem initialization string, not including the “AT”. If not specified (valueLen of 0), the modem initialization string from system preferences are used.</td>
</tr>
<tr>
<td>ModemPhone</td>
<td>Char[]</td>
<td>Zero-terminated modem phone number string. Only required if netIFSettingUseModem is true.</td>
</tr>
</tbody>
</table>
### Table 61.2 Network Interface Settings (continued)

<table>
<thead>
<tr>
<th>netIFSetting...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RedialCount</td>
<td>UInt16</td>
<td>Number of times to re-dial modem when trying to establish a connection. Only required if netIFSettingUseModem is true.</td>
</tr>
<tr>
<td>DNSQuery</td>
<td>UInt8</td>
<td>true if PPP queries for DNS address. The default is true.</td>
</tr>
<tr>
<td>TraceBits</td>
<td>UInt32</td>
<td>A bitfield of various trace bits. See &quot;Tracing Bits.&quot; Each interface has its own trace bits setting so that trace event recording in each interface can be selectively enabled or disabled.</td>
</tr>
<tr>
<td>ActualIPAddr</td>
<td>UInt32</td>
<td>Read-only. The actual IP address that the interface ends up using. The login script execution engine stores the result of the &quot;g&quot; (get IP address) command here as does the PPP negotiation logic.</td>
</tr>
<tr>
<td>ServerIPAddr</td>
<td>UInt32</td>
<td>Read-only. The IP address of the PPP server we’re connected to.</td>
</tr>
<tr>
<td>BringDownOnPowerDown</td>
<td>UInt8</td>
<td>true if the interface is brought down when the Palm OS device is turned off.</td>
</tr>
<tr>
<td>RawMode</td>
<td>UInt32</td>
<td>Specifies if the interface is in raw mode. The net library places an interface in raw mode when it is bound to a raw socket in the raw domain. Raw sockets are available in Palm OS version 3.0 and higher.</td>
</tr>
<tr>
<td>DriverVersion</td>
<td>Char[20]</td>
<td>Read-only. The version number of the network interface device driver. This setting is defined only if 5.1 New Feature Set is present.</td>
</tr>
<tr>
<td>FirmwareVersion</td>
<td>Char[20]</td>
<td>Read-only. The firmware version of the network interface device, if any. This setting is defined only if 5.1 New Feature Set is present.</td>
</tr>
</tbody>
</table>
Table 61.2 Network Interface Settings (continued)

<table>
<thead>
<tr>
<th>netIFSetting...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FirmwareDate</td>
<td>UInt32</td>
<td>Read-only. Firmware date in seconds since midnight, January 1, 1904. This setting is defined only if 5.1 New Feature Set is present.</td>
</tr>
<tr>
<td>80211Device</td>
<td>UInt8</td>
<td>Read-only. Indicates whether or not the interface supports IEEE 802.11 wireless networking. This setting is defined only if 5.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New Feature Set is present.</td>
</tr>
<tr>
<td>80211ESSID</td>
<td>Char[32]</td>
<td>For IEEE 802.11 interfaces only. The ESS ID of the radio. This setting is defined only if 5.1 New Feature Set is present.</td>
</tr>
<tr>
<td>80211AccessPointBSSID</td>
<td>UInt8[6]</td>
<td>Read-only. For IEEE 802.11 interfaces only. The BSS ID (MAC address) of the access point to which the radio is connected. This setting is defined only if 5.1 New Feature Set is present.</td>
</tr>
<tr>
<td>80211AssociationStatus</td>
<td>UInt8</td>
<td>Read-only. For IEEE 802.11 interfaces only. true if the radio is associated. This setting is defined only if 5.1 New Feature Set is present.</td>
</tr>
<tr>
<td>80211MKKCallSign</td>
<td>Char[15]</td>
<td>Read-only. For IEEE 802.11 interfaces with radios programmed for operation in Japan only. The MKK call sign. This setting is defined only if 5.1 New Feature Set is present.</td>
</tr>
<tr>
<td>80211CountryText</td>
<td>Char[34]</td>
<td>Read-only. For IEEE 802.11 interfaces only. The radio’s country code, which the radio uses to check if it operates within a particular country’s regulations. This setting is defined only if 5.1 New Feature Set is present.</td>
</tr>
</tbody>
</table>

**See Also** NetLibIFSettingSet, NetLibSettingGet, NetLibSettingSet, “Interface Specific Settings” in the Palm OS Programmer’s Companion, vol. II, Communications
NetLibIFSettingSet

**Purpose** Sets a network interface specific setting.

**Declared In** NetMgr.h

**Prototype**

```c
Err NetLibIFSettingSet (UInt16 libRefNum,
UInt32 ifCreator, UInt16 ifInstance,
UInt16 setting, void *valueP, UInt16 valueLen)
```

**Parameters**

- **libRefNum** Reference number of the net library.
- **ifCreator** Creator of the network interface.
- **ifInstance** Instance number of the network interface.
- **setting** The setting to set, one of the NetIFSettingEnum constants. See Table 61.2.
- **valueP** Space new value of setting.
- **valueLen** Size of new setting.

**Result**

Returns one of the following values:

- 0 Success.
- netErrUnknownSetting Invalid setting constant.
- netErrPrefNotFound No current value for setting.
- netErrUnimplemented
- netErrInterfaceNotFound
- netErrBufWrongSize
- netErrReadOnlySetting

**Sockets Equivalent**

None

**Comments**

This call can be used to set the current value of any network interface setting. The caller must pass a pointer to a buffer which
holds the new value (valueP), the size of the buffer (valueLen), and the setting ID (setting).

See NetLibIFSettingGet for an explanation of each of the settings.

Of particular interest is the netIFSettingResetAll setting, which, if used, resets all settings for the interface to their default values. When using this setting, valueP and valueLen are ignored.

See Also NetLibIFSettingGet, NetLibSettingGet, NetLibSettingSet, “Interface Specific Settings” in the Palm OS Programmer’s Companion, vol. II, Communications

NetLibIFUp

Purpose Bring an interface up and establish a connection.

Declared In NetMgr.h

Prototype Err NetLibIFUp (UInt16 libRefNum,
UInt32 ifCreator, UInt16 ifInstance)

Parameters
-► libRefNum Reference number of the net library.
-► ifCreator Creator of interface to attach.
-► ifInstance Instance number of interface to attach.

Result Returns one of the following values:

0 Success.

netErrNotOpen The referenced net library has not been opened yet.

netErrInterfaceNotFound
netErrUserCancel
netErrBadScript
netErrPPPTimeout
Net Library
Net Library Functions

netErrAuthFailure
netErrPPPAddressRefused

Sockets
Equivalent
None

Comments
The net library must be open before this call can be made. For dial-up interfaces, this call will dial up the modem if necessary and run through the connect script to establish the connection.

If the interface is already up, this routine returns immediately with no error. This call doesn’t take a timeout parameter because it relies on each interface to have its own established timeout setting.

See Also
NetLibIFGet, NetLibIFAttach, NetLibIFDetach, NetLibIFDown

NetLibMaster

Purpose
Retrieves the network statistics, interface statistics, and the contents of the trace buffer.

Declared In
NetMgr.h

Prototype
Err NetLibMaster (UInt16 libRefNum, UInt16 cmd,
NetMasterPBPtr pbP, Int32 timeout)

Parameters
-> libRefNum Reference number of the net library.
-> cmd Function to perform (NetMasterEnum type).
The following commands are supported:
  netMasterInterfaceInfo
  netMasterInterfaceStats
  netMasterIPStats
  netMasterICMPStats
  netMasterUDPStats
  netMasterTCPStats
  netMasterTraceEventGet
<-> pbP Command parameter block.
Net Library
Net Library Functions

-> timeout  Timeout in ticks; -1 means wait forever.

Result
Returns one of the following values:

0  No error
netErrNotOpen  The referenced net library has not been opened yet.
netErrParamErr
netErrUnimplemented

Sockets
Equivalent
None

Comments
This call allows applications to get detailed information about the net library. This information is usually helpful in debugging network configuration problems.

This function takes a command word (cmd) and parameter block pointer (pbP) as arguments and returns its results in the parameter block on exit. Which values you must specify in the parameter block and which values are returned are specific to the command you specify.

netMasterInterfaceInfo
The pbP->interfaceInfo struct specifies interface information.

-> index  Index of interface to fetch info about.
<- creator  Creator of interface.
<- instance  Instance of interface.
<- netIFP  Private interface info pointer.
<- drvrName  Driver type that interface uses (“PPP”, “SLIP”, etc.).
<- hwName  Hardware driver name (“Serial Library”, etc.).
### netMasterInterfaceStats

The `pbP->interfaceStats` structure specifies interface statistics.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>index</code></td>
<td>Index of interface to fetch info about.</td>
</tr>
<tr>
<td><code>inOctets</code></td>
<td>Number of octets received.</td>
</tr>
<tr>
<td><code>inUcastPkts</code></td>
<td>Number of packets received.</td>
</tr>
<tr>
<td><code>inNUcastPkts</code></td>
<td>Number of broadcast packets received.</td>
</tr>
<tr>
<td><code>inDiscards</code></td>
<td>Number of incoming packets that were discarded.</td>
</tr>
</tbody>
</table>

### Local Net Interface Stats

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>localNetHdrLen</code></td>
<td>Number of bytes in local net header.</td>
</tr>
<tr>
<td><code>localNetTrailerLen</code></td>
<td>Number of bytes in local net trailer.</td>
</tr>
<tr>
<td><code>localNetMaxFrame</code></td>
<td>Local net maximum frame size.</td>
</tr>
<tr>
<td><code>ifName</code></td>
<td>Interface name with instance number concatenated.</td>
</tr>
<tr>
<td><code>driverUp</code></td>
<td>True if interface driver is up.</td>
</tr>
<tr>
<td><code>ifUp</code></td>
<td>True if interface media layer is up.</td>
</tr>
<tr>
<td><code>hwAddrLen</code></td>
<td>Length of interface’s hardware address.</td>
</tr>
<tr>
<td><code>hwAddr</code></td>
<td>Interface’s hardware address.</td>
</tr>
<tr>
<td><code>mtu</code></td>
<td>Maximum transfer unit of interface.</td>
</tr>
<tr>
<td><code>speed</code></td>
<td>Speed in bits per second.</td>
</tr>
<tr>
<td><code>lastStateChange</code></td>
<td>Time in milliseconds of last state change.</td>
</tr>
<tr>
<td><code>ipAddr</code></td>
<td>IP address of interface.</td>
</tr>
<tr>
<td><code>subnetMask</code></td>
<td>Subnet mask of local network.</td>
</tr>
<tr>
<td><code>broadcast</code></td>
<td>Broadcast address of local network.</td>
</tr>
</tbody>
</table>
netMasterIPStats
The pbP->ipStats structure contains statistics about the IP protocol. See NetMgr.h for a complete list of statistics returned.

netMasterICMPStats
The pbP->icmpStats structure contains statistics about the ICMP protocol. See NetMgr.h for a complete list of statistics returned.

netMasterUDPStats
The pbP->udpStats structure contains statistics about the UDP protocol. See NetMgr.h for a complete list of statistics returned.

netMasterTCPStats
The pbP->tcpStats structure contains statistics about the TCP protocol. See NetMgr.h for a complete list of statistics returned.

netMasterTraceEventGet
The pbP->traceEventGet structure contains a trace event.

<- inErrors Number of packet errors encountered.
<- inUnknownProtos Number of unknown protocols encountered.
<- outOctets Number octets sent.
<- outUcastPkts Number of packets sent.
<- outNUcastPkts Number of broadcast packets sent.
<- outDiscards Number of packets discarded.
<- outErrors Number of outbound packet errors.

See Also NetLibSettingSet
NetLibOpen

Purpose
Opens and initializes the net library.

Declared In
NetMgr.h

Prototype
Err NetLibOpen (UInt16 libRefnum, UInt16 *netIFErrsP)

Parameters
- libRefnum Reference number of the net library.
- netIFErrsP First error encountered when bringing up network interfaces. (See NetLibIFUp for a list of possible values.)

Result
Returns one of the following values:

- 0 No error.
- netErrAlreadyOpen Not really an error; returned if library was already open and the open count was simply incremented.
- netErrOutOfMemory Not enough memory available to open the library.
- netErrNoInterfaces Incorrect setup.
- netErrPrefNotFound Incorrect setup.

Comments
Applications must call this function before using the net library. If the net library was already open, NetLibOpen increments its open count. Otherwise, it opens the library, initializes it, starts up the net protocol stack component of the library as a separate task, and brings up all attached network interfaces.

NetLibOpen uses settings saved in the net library’s preferences database during initialization. These settings include the interfaces to attach, the IP addresses, etc. It’s assumed that these settings have
been previously set up by a preference panel or equivalent so an application doesn’t normally have to set them up before calling NetLibOpen.

If any of the attached interfaces fails to come up, *netIFErrsP will contain the error number of the first interface that encountered a problem.

**Compatibility**

NetLibOpen behaves slightly differently in version 3.2 and later than it does in previous releases. In version 3.2 and later, NetLibOpen calls NetLibOpenConfig specifying the default configuration. NetLibOpenConfig reverts all settings to their saved, default values before opening the net library.

**See Also**

SysLibFind, NetLibClose, NetLibOpenCount

---

**New**

NetLibOpenConfig

**Purpose**

Opens and initializes the net library with the specified configuration.

**Prototype**

Err NetLibOpenConfig (UInt16 refNum, UInt16 configIndex, UInt32 openFlags, UInt16 *netIFErrP)

**Parameters**

- `-> refNum` Reference number of the net library.
- `-> configIndex` Index of the configuration to use. 0 means use the default configuration as defined in the Network preferences panel.
- `-> openFlags` Not used. Pass 0 for this parameter.
- `<- netIFErrP` Pointer to return error code for interfaces.

**Result**

Returns one of the following values:

0 No error.

memErrNotEnoughSpace
netErrAlreadyOpen
Not really an error; returned if library was already open and the open count was simply incremented.

netErrAlreadyOpenWithAnotherConfig
Another application has the net library open with a configuration that is incompatible with the one specified.

netErrBufTooSmall

netErrConfigAliasErr
A configuration alias was specified, but the alias could not be resolved.

netErrConfigCantDelete
netErrConfigEmpty
The configuration is not defined.

netErrConfigNotFound
The specified configuration index is invalid.

netErrInterfaceNotFound
netErrOutOfCmdBlocks
netErrOutOfMemory
Not enough memory available to open the library.

netErrNoInterfaces
Incorrect setup.

netErrParamErr
netErrPrefNotFound
Incorrect setup.

netErrTimeout

**Sockets Equivalent**
None

**Comments**
Use this routine instead of [NetLibOpen](#) when you want to open the net library with a non-default configuration. If the default net library configuration is not suitable for your application, you may
use one of the built-in aliases to specify a configuration that is suitable (see “Configuration Aliases”).

NetLibOpenConfig tries to open the net library and initialize it with the specified configuration. If another application has the net library open with an incompatible configuration, it returns an error. If the net library is in the close-wait state, this function completely closes the net library and then reopens it using the new configuration. If the net library can be opened with the new configuration, NetLibOpenConfig first saves the current configuration so that it can be restored when your application closes the net library.

Typically, applications use the NetLibConfigList function to obtain the list of available configurations and present this list to the user. Then they call NetLibConfigIndexFromName with the user’s selection to get the index of the configuration that the user selected.

The constant netConfigIndexCurSettings specifies the current configuration. You can specify netConfigIndexCurSettings as the configIndex for testing purposes.

**Compatibility**

Implemented only if 3.2 New Feature Set is present.

**See Also**

NetLibOpen, SysLibFind, NetLibClose, NetLibOpenCount

### NetLibOpenCount

**Purpose**

Retrieves the open count of the net library.

**Declared In**

NetMgr.h

**Prototype**

Err NetLibOpenCount (UInt16 refNum, UInt16 *countP)

**Parameters**

- `-> refNum` Reference number of the net library.
<- countP Contains the open count of the net library upon return.

Result Always returns 0.

Sockets Equivalent None.

Comments This call will most likely only be used by the Network preferences panel. Most applications will simply call NetLibOpen unconditionally during startup and NetLibClose when they exit.

**NetLibReceive**

**Purpose** Receive data from a socket into a single buffer.

**Declared In** NetMgr.h

**Prototype**

```c
Int16 NetLibReceive (UInt16 libRefNum,
NetSocketRef socket, void *bufP, UInt16 bufLen,
UInt16 flags, void *fromAddrP, UInt16 *fromLenP,
Int32 timeout, Err *errP)
```

**Parameters**

- `-> libRefNum` Reference number of the net library.
- `-> socket` Descriptor for the open socket.
- `<- bufP` Pointer to buffer to hold received data.
- `<- bufLen` Length of `bufP` buffer.
- `<- flags` One or more netIOFlagxxx flags. See “I/O Flags.”
- `<- fromAddrP` Pointer to buffer to hold address of sender (a NetSocketAddrType).
- `<> fromLenP` On entry, size of `fromAddrP` buffer. On exit, actual size of returned address in `fromAddrP`.
- `-> timeout` Maximum timeout in system ticks; -1 means wait forever.
<-errP Contains an error code if the return value is -1.

**Result**
Returns the number of bytes successfully received. If the return value is 0, the socket has been shut down by the remote host. If the return value is -1, an error has occurred, and errP contains one of the following values:

- 0 No error.
- netErrTimeout Call timed out.
- netErrNotOpen The referenced net library has not been opened yet.
- netErrParamErr
- netErrSocketNotOpen
- netErrWouldBlock
- netErrUserCancel

**Sockets Equivalent**

```c
int recvfrom (int socket, const void *bufP, int bufLen, int flags, const void *fromAddrP, int *fromLenP);
int recv (int socket, const void *bufP, int bufLen, int flags);
int read (int socket, const void *bufP, int bufLen);
```

**Comments**
For stream-based sockets, this call reads whatever bytes are available and returns the number of bytes actually read into the caller’s buffer. If there is no data available, this call will block until at least one byte arrives, until the socket is shut down by the remote host, or until a timeout occurs.

For datagram-based sockets, this call reads a complete datagram and returns the number of bytes in the datagram. If the caller’s buffer is not large enough to hold the entire datagram, the end of the datagram is discarded. If a datagram is not available, this call will block until one arrives, or until the call times out.
The data is read into a single buffer pointed to by bufP.

See Also NetLibReceive, NetLibDmReceive, NetUReadN, NetLibSend, NetLibSendPB

**NetLibReceivePB**

**Purpose** Receive data from a socket into a multi-buffer gather-read array.

**Declared In** NetMgr.h

**Prototype**

```
Int16 NetLibReceivePB (UInt16 libRefNum, NetSocketRef socket, NetIOParamType *pbP, UInt16 flags, Int32 timeout, Err *errP)
```

**Parameters**

- `libRefNum` Reference number of the net library.
- `socket` Descriptor for the open socket.
- `pbP` Pointer to parameter block containing buffer info.
- `flags` One or more netIOFlagxxx flags. See “[I/O Flags](#)”
- `timeout` Maximum timeout in system ticks; -1 means wait forever.
- `errP` Contains an error code if the return value is -1.

**Result** Returns the number of bytes successfully received. Returns 0 if the socket has been shut down by the remote host. If the return value is -1, an error has occurred, and `errP` contains one of the following values:

- `0` No error.
- `netErrTimeout` Call timed out.
- `netErrNotOpen` The referenced net library has not been opened yet.
- `netErrParamErr`
- `netErrSocketNotOpen`
netErrWouldBlock

Sockets Equivalent

```c
int recvmsg (int socket, const struct msghdr *pbP, int flags);
```

Comments

The `pbP` parameter is a pointer to a `NetIOParamType` structure. `NetIOParamType` is defined as follows:

```c
typedef struct {
    UInt8    *addrP;
    UInt16   addrLen;
    NetIOVecPtr iov;
    UInt16   iovLen;
    UInt8    *accessRights;
    UInt16   accessRightsLen;
} NetIOParamType, *NetIOParamPtr;
```

You provide the following information in this struct:

- **addrP**: Address of sender, set by `NetLibReceivePB`. Set to 0 if you don’t require this field.
- **addrLen**: Length of `*addrP`.
- **iov**: Array of buffers into which the data should be received. `NetIOVecPtr` is a pointer to a `NetIOVecType` structure, which has two fields:
  - `bufP`: Pointer to a buffer.
  - `bufLen`: Length of `bufP`.
- **iovLen**: Length of the `iov` array.
- **accessRights**: Access rights. This field currently isn’t used and should be set to 0.
- **accessRightsLen**: Length of the `*accessRights`. This field currently isn’t used and should be set to 0.

For stream-based sockets, this call reads whatever bytes are available and returns the number of bytes actually read into the caller’s buffer. If no data is available, this call will block until at least
one byte arrives, until the socket is shut down by the remote host, or until a timeout occurs.

For datagram-based sockets, this call reads a complete datagram and returns the number of bytes in the datagram. If the caller’s buffer is not large enough to hold the entire datagram, the end of the datagram is discarded. If a datagram is not available, this call will block until one arrives, or until the call times out.

The data is read into the gather-read array specified by the pbP->iov array.

See Also NetLibReceive, NetLibDmReceive, NetLibSend, NetLibSendPB

NetLibSelect

Purpose Blocks until I/O is ready on one or more descriptors, where a descriptor can represent socket input, socket output, or a user input event like a pen tap or key press.

Declared In NetMgr.h

Prototype Int16 NetLibSelect (UInt16 libRefNum, UInt16 width, NetFDSetType *readFDs, NetFDSetType *writeFDs, NetFDSetType *exceptFDs, Int32 timeout, Err *errP)

Parameters -> libRefNum Reference number of the net library.
-> width Number of descriptor bits to check in the readFDs, writeFDs, and exceptFDs descriptor sets.
<> readFDs Pointer to 32-bit NetFDSetType containing set of bits representing descriptors to check for input.
<> writeFDs Pointer to 32-bit NetFDSetType containing set of bits representing descriptors to check for output.
Net Library
Net Library Functions

exceptFDs  Pointer to 32-bit NetFDSetType containing set of bits representing descriptors to check for exception conditions. This parameter is ignored. Upon return, its bits are always cleared.

timeout  Maximum timeout in system ticks; -1 means wait forever.

errP  Contains an error code if the return value is -1.

Result  Returns the sum total number of ready file descriptors in *readFDs, *writeFDs, and *exceptFDs. Returns 0 upon timeout. If the return value is -1, an error has occurred, and errP contains one of the following values:

0  No error
netErrTimeout  Call timed out.
netErrNotOpen  The referenced net library has not been opened yet.

Sockets Equivalent

int select (int width, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);

Comments  This call blocks until one or more descriptors are ready for I/O. In the Palm OS environment, a descriptor is either a NetSocketRef or the “stdin” descriptor, sysFileDescStdIn. The sysFileDescStdIn descriptor will be ready for input whenever a user event is available like a pen tap or key press.

The caller should set which bits in each descriptor set need to be checked by using the netFDZero and netFDSet macros. After this call returns, the macro netFDIsSet can be used to determine which descriptors in each set are actually ready.

On exit, the total number of ready descriptors is returned and each descriptor set is updated with the appropriate bits set for each ready descriptor in that set.

The following example illustrates how to use this call to check for input on a socket or a user event:
Err    err;
NetSocketRef  socket;
NetFDSetType   readFDs,writeFDs,exceptFDs;
Int16      numFDs;
UInt16     width;

// Create the descriptor sets
netFDZero(&readFDs);
netFDZero(&writeFDs);
netFDZero(&exceptFDs);
netFDSet(sysFileDescStdIn, &readFDs);
netFDSet(socket, &readFDs);

// Calculate the max descriptor number and
// use that +1 as the max width.
// Alternatively, we could simply use the
// constant netFDSetSize as the width which
// is simpler but makes the NetLibSelect call
// slightly slower.
width = sysFileDescStdIn;
if (socket > width) width = socket;

// Wait for any one of the descriptors to be
// ready.
numFDs = NetLibSelect(AppNetRefnum, width+1,
                     &readFDs, &writeFDs, &exceptFDs,
                     AppNetTimeout, &err);

Also see the NetSample example application in the Palm OS
Examples folder. The function CmdTelnet in the file
CmdTelnet.c shows how to use the Berkeley sockets select
function and how to interpret the results.

See Also  NetLibSocketOptionSet
Net Library
Net Library Functions

NetLibSend

Purpose
Send data to a socket from a single buffer.

Declared In
NetMgr.h

Prototype
Int16 NetLibSend (UInt16 libRefNum,
NetSocketRef socket, void *bufP, UInt16 bufLen,
UInt16 flags, void *toAddrP, UInt16 toLen,
Int32 timeout, Err *errP)

Parameters
- `libRefNum` Reference number of the net library.
- `socket` Descriptor for the open socket.
- `bufP` Pointer to data to write.
- `bufLen` Length of data to write
- `flags` One or more netIOFlagxxx flags. See "I/O Flags."
- `toAddrP` Address to send to (a pointer to a NetSocketAddrType), or 0.
- `toLen` Size of toAddrP buffer.
- `timeout` Maximum timeout in system ticks; -1 means wait forever.
- `errP` Contains an error code if the return value is -1.

Result
Returns the number of bytes successfully sent. Returns 0 if the socket has been shut down by the remote host. If the return value is -1, an error has occurred, and errP contains one of the following values:

- 0 No error.
- netErrTimeout Call timed out.
- netErrNotOpen The referenced net library has not been opened yet.
- netErrParamErr
- netErrSocketNotOpen

1474  Palm OS Programmer’s API Reference
netErrMessageTooBig
netErrSocketNotConnected
netErrSocketClosedByRemote
netErrIPCantFragment
netErrIPNoRoute
netErrIPNoSrc
netErrIPNoDst
netErrIPktOverflow
netErrOutOfCmdBlocks
netErrOutOfPackets
netErrInterfaceNotFound
netErrInterfaceDown
netErrUnreachableDest
netErrNoMultiPktAddr
netErrWouldBlock

Sockets Equivalent

```c
int sendto (int socket, const void *bufP,
           int bufLen, int flags, const void *toAddrP,
           int toLen);
```

```c
int send (int socket, const void *bufP,
         int bufLen, int flags);
```

```c
int write (int socket, const void *bufP,
          int bufLen,);
```

Comments

This call attempts to write data to the specified socket and returns the number of bytes actually sent, which may be less than or equal to the requested number of bytes. The data is passed in a single buffer that bufP points to.

For datagram sockets, you must only send a single packet at a time. If the data is too large to fit in a single UDP packet (1536 bytes), no data is sent and -1 is returned.
The `toAddrP` field applies only to datagram sockets without an existing connection. An error is returned if the datagram socket was previously connected and `toAddrP` is specified. Stream-based sockets, by definition, must have a connection established with a remote host before data can be written. Raw sockets (supported in Palm OS version 3.0 and higher) must construct the entire IP header, including the destination address, before data can be sent; thus, the address is taken from the data to be sent.

If there isn’t enough buffer space to send any data, this call will block until there is enough buffer space, or until a timeout.

**NOTE:** For stream-based sockets, this call may write only a portion of the desired data. It always returns the number of bytes actually written. Consequently, the caller should be prepared to call this routine repeatedly until the desired number of bytes have been written, or until it returns 0 or -1.

### See Also

NetLibSendPB, NetUWriteN, NetLibReceive, NetLibReceivePB, NetLibDmReceive

## NetLibSendPB

### Purpose
Send data to a socket from a scatter-write array.

### Declared In
NetMgr.h

### Prototype

```
Int16 NetLibSendPB (UInt16 libRefNum,
                     NetSocketRef socket, NetIOParamType *pbP,
                     UInt16 flags, Int32 timeout, Err *errP)
```

### Parameters

- `-> libRefNum` Reference number of the net library.
- `-> socket` Descriptor for the open socket.
- `-> pbP` Pointer to parameter block containing buffer info. See the description in NetLibReceivePB.
Net Library
Net Library Functions

> flags One or more netIOFlagxxx flags. See "I/O Flags."
> timeout Maximum timeout in system ticks; -1 means wait forever.
<- errP Contains an error code if the return value is -1.

Result Returns the number of bytes successfully sent. Returns 0 if the socket has been shut down by the remote host. If the return value is -1, an error has occurred, and errP contains one of the following values:

0 No error.
```
netErrTimeout Call timed out.
netErrNotOpen The referenced net library has not been opened yet.
netErrParamErr
netErrSocketNotOpen
netErrMessageTooBig
netErrSocketNotConnected
netErrSocketClosedByRemote
netErrIPCantFragment
netErrIPNoRoute
netErrIPNoSrc
netErrIPNoDst
netErrIPktOverflow
netErrOutOfCmdBlocks
netErrOutOfPackets
netErrInterfaceNotFound
netErrInterfaceDown
netErrUnreachableDest
netErrNoMultiPktAddr
netErrWouldBlock
```
Sockets Equivalent

```c
int sendmsg (int socket, const struct msghdr *pbP, int flags);
```

Comments

This call attempts to write data to the given socket and returns the number of bytes actually sent, which may be less than or equal to the requested number of bytes. The data is passed in the scatter-write array specified in the `pbP` parameter block.

For datagram sockets, you must only send a single packet at a time. If the data is too large to fit in a single UDP packet, no data is sent and -1 is returned.

The `toAddrP` field applies only to datagram sockets without an existing connection. An error is returned if the datagram socket was previously connected and `toAddrP` is specified. Stream-based sockets, by definition, must have a connection established with a remote host before data can be written. Raw sockets (supported in Palm OS version 3.0 and higher) must construct the entire IP header, including the destination address, before data can be sent; thus, the address is taken from the data to be sent.

If there isn’t enough buffer space to send any data, this call will block until there is space, or until a timeout.

**NOTE:** For stream-based sockets, this call may write only a portion of the desired data. It always returns the number of bytes actually written. Consequently, the caller should be prepared to call this routine repeatedly until the desired number of bytes have been written, or until it returns 0 or -1.

See Also

- `NetLibSend`
- `NetLibReceive`
- `NetLibReceivePB`
- `NetLibDmReceive`
**NetLibSettingGet**

**Purpose**
Retrieves a general setting.

**Declared In**
NetMgr.h

**Prototype**
```
Err NetLibSettingGet (UInt16 libRefNum,
        UInt16 setting, void *valueP, UInt16 *valueLenP)
```

**Parameters**
- `-> libRefNum` Reference number of the net library.
- `-> setting` Setting to retrieve, one of the NetSettingEnum constants.
- `<- valueP` Space for return value of setting.
- `<-> valueLenP` On entry, size of `valueP`. On exit, actual size of setting.

**Result**
Returns one of the following values:
- `0` Success.
- `netErrUnknownSetting` Invalid setting constant
- `netErrPrefNotFound` No current value for setting
- `netErrBufTooSmall` `valueP` was too small to hold entire setting. Setting value was truncated to fit in `valueP`.
- `netErrBufWrongSize` None

**Comments**
This call retrieves the current value of any general setting. The caller must pass a pointer to a buffer to hold the return value (`valueP`), the size of the buffer (`valueLenP`), and the setting ID (`setting`). The setting ID is one of the NetSettingEnum constants in the NetSettingEnum type.
Some settings, such as the host table, are variable size. For these types of settings, you can obtain the actual size required for the buffer by passing 0 for *valueLenP. The required size is returned in valueLenP.

Table 61.3 lists the general settings and the type of each setting.
## Table 61.3 Net Library General Settings

<table>
<thead>
<tr>
<th>netSetting...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResetAll</td>
<td>void</td>
<td>Used for NetLibSettingSet only. This will clear all other settings to their default values.</td>
</tr>
<tr>
<td>PrimaryDNS</td>
<td>UInt32</td>
<td>IP address of primary DNS server. This setting must be set to a non-zero IP address in order to support any of the name lookup calls.</td>
</tr>
<tr>
<td>SecondaryDNS</td>
<td>UInt32</td>
<td>IP address of secondary DNS server. Set to 0 to have stack ignore this setting.</td>
</tr>
<tr>
<td>DefaultRouter</td>
<td>UInt32</td>
<td>IP address of default router. Default value is 0 which is appropriate for most implementations with only one attached interface (besides loopback). Packets with destination IP addresses that don’t lie in the subnet of an attached interface will be sent to this router through the default interface specified by the netSettingDefaultIFCreator/ netSettingDefaultIFInstance pair.</td>
</tr>
<tr>
<td>DefaultIFCreator</td>
<td>UInt32</td>
<td>Creator of the default network interface. Default value is 0, which is appropriate for most implementations. Packets with destination IP addresses that don’t lie in the subnet of a directly attached interface are sent through this interface. If this setting is 0, the stack automatically makes the first non-loopback interface the default interface.</td>
</tr>
<tr>
<td>DefaultIFInstance</td>
<td>UInt16</td>
<td>Instance number of the default network interface. Packets with destination IP addresses that don’t lie in the subnet of an attached interface are sent through the default interface. Default value is 0.</td>
</tr>
<tr>
<td>HostName</td>
<td>Char[]</td>
<td>A zero-terminated character string of 64 bytes or less containing the host name of this machine. This setting is not actually used by the stack. It’s present mainly for informative purposes and to support the gethostname/sethostname sockets API calls. To clear the host name, call NetLibIFSettingSet with a valueLen of 0.</td>
</tr>
</tbody>
</table>
### Net Library Functions

**See Also** \[NetLibSettingSet, NetLibIFSettingSet, NetLibIFSettingGet, NetLibMaster\]

---

**Table 61.3 Net Library General Settings (continued)**

<table>
<thead>
<tr>
<th>netSetting...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DomainName</td>
<td>Char []</td>
<td>A zero-terminated character string of 256 bytes or less containing the default domain. This default domain name is appended to all host names before name lookups are performed. If the name is not found, the host name is looked up again without appending the domain name to it. To have the stack not use the domain name, call NetLibIFSettingSet with a valueLen of 0.</td>
</tr>
<tr>
<td>HostTbl</td>
<td>Char []</td>
<td>A null-terminated character string containing the host table. This table is consulted first before sending a DNS query to the DNS server(s). To have the stack not use a host table, call NetLibIFSettingSet with a valueLen of 0. The format of a host table is a series of lines separated by ‘\n’ in the following format:</td>
</tr>
<tr>
<td>CloseWaitTime</td>
<td>UInt32</td>
<td>The close-wait time in milliseconds. This setting must be specified. See the discussion of the NetLibClose call for an explanation of the close-wait time.</td>
</tr>
<tr>
<td>TraceBits</td>
<td>UInt32</td>
<td>A bitfield of various trace bits. See “Tracing Bits.” Default value is (netTracingErrors</td>
</tr>
<tr>
<td>TraceSize</td>
<td>UInt32</td>
<td>Maximum trace buffer size in bytes. Setting this setting always clears the existing trace buffer. Default is 2 KB.</td>
</tr>
<tr>
<td>TraceRoll</td>
<td>UInt8</td>
<td>Boolean value, default is true (non-zero). If true, trace buffer will roll over when it fills. If false, tracing will stop as soon as trace buffer fills.</td>
</tr>
</tbody>
</table>
**NetLibSettingSet**

**Purpose**
Sets a general setting.

**Declared In**
NetMgr.h

**Prototype**
```c
Err NetLibSettingSet (UInt16 libRefNum,
                      UInt16 setting, void *valueP, UInt16 valueLen)
```

**Parameters**
- `libRefNum`  Reference number of the net library.
- `setting`  Setting to set; one of the `NetSettingEnum` constants. See Table 61.3.
- `valueP`  New value for the setting.
- `valueLen`  Size of new setting.

**Result**
Returns one of the following values:
- 0  Success.
- `netErrUnknownSetting`  Invalid setting constant.
- `netErrInvalidSettingSize`  `valueLen` was invalid for the given setting.
- `netErrBufWrongSize`
- `netErrReadOnlySetting`

**Sockets Equivalent**
None

**Comments**
This call can be used to set the current value of any general setting. The caller must pass a pointer to a buffer which holds the new value (`valueP`), the size of the buffer (`valueLen`), and the setting ID (`setting`). The setting ID is one of the `netSettingXXX` constants in the `NetSettingEnum` type.

If the net library is not open at the time this call is made, the setting is stored in the active configuration. You need to save the active configuration using `NetLibConfigSaveAs` if you want the new
value of the setting to be used the next time the net library is
opened.
See NetLibSettingGet for an explanation of each of the settings.
Of particular interest is the netSettingResetAll setting, which,
if used, will reset all general settings to their default values. When
using this setting, valueP and valueLen are ignored.

See Also NetLibSettingGet, NetLibSettingSet,
NetLibIPSettingSet, NetLibMaster

NetLibSocketAccept

Purpose Accept a connection from a stream-based socket.

Declared In NetMgr.h

Prototype Int16 NetLibSocketAccept (UInt16 libRefnum,
NetSocketRef socket,
NetSocketAddrType *sockAddrP, Int16 *addrLenP,
Int32 timeout, Err *errP)

Parameters
- libRefNum Reference number of the net library.
- socket Descriptor for the open socket.
- sockAddrP Address of remote host is returned here.
- addrLenP On entry, length of sockAddrP buffer in bytes. On
  exit, length of returned address stored in *sockAddrP.
- timeout Maximum timeout in system ticks; -1 means
  wait forever.
- errP Contains an error code if the return value is -1.

Result Returns the NetSocketRef of the new socket. If the return value is
-1, an error has occurred, and errP contains one of the following
values:
0 No error.
netErrTimeout  Call timed out.
netErrNotOpen  The referenced net library has not been opened yet.
netErrParamErr
netErrSocketNotOpen
netErrSocketNotConnected
netErrSocketClosedByRemote
netErrWrongSocketType
netErrSocketNotListening
netErrUnimplemented

### Sockets Equivalent

```c
int accept (int socket, void *sockAddrP, int *addrLenP);
```

### Comments

Accepts the next connection request from a remote client. This call is only applicable to stream-based sockets. Before calling `NetLibSocketAccept` on a socket, a server application needs to:

- Open the socket (`NetLibSocketOpen`).
- Bind the socket to a local address (`NetLibSocketBind`).
- Set the maximum pending connection-request queue length (`NetLibSocketListen`).

`NetLibSocketAccept` will block until a successful connection request is obtained from a remote client. After a successful connection is made, this call returns with the address of the remote host in `sockAddrP` and the socket descriptor of a new socket as the return value. You then use the new socket to send and receive data.

### See Also

`NetLibSocketBind`, `NetLibSocketListen`
Net Library
Net Library Functions

NetLibSocketAddr

**Purpose**
Returns the local and remote addresses currently associated with a socket.

**Declared In**
NetMgr.h

**Prototype**
```c
Int16 NetLibSocketAddr (UInt16 libRefnum,
    NetSocketRef socketRef,
    NetSocketAddrType *locAddrP, Int16 *locAddrLenP,
    NetSocketAddrType *remAddrP, Int16 *remAddrLenP,
    Int32 timeout, Err *errP)
```

**Parameters**
- **libRefNum**
  Reference number of the net library.
- **socketRef**
  Descriptor for the open socket.
- **locAddrP**
  Local address of socket is returned here.
- **locAddrLenP**
  On entry, length of `locAddrP` buffer in bytes. On exit, length of returned address stored in `*locAddrP`.
- **remAddrP**
  Address of remote host is returned here.
- **remAddrLenP**
  On entry, length of `remAddrP` buffer in bytes. On exit, length of returned address stored in `*remAddrP`.
- **timeout**
  Maximum timeout in system ticks; -1 means wait forever.
- **errP**
  Contains an error code if the return value is -1.

**Result**
Returns 0 upon success and -1 if an error occurred. If the return value is -1, `errP` contains one of the following values:

- 0: No error.
- netErrTimeout: Call timed out.
- netErrNotOpen: The referenced net library has not been opened yet.
- netErrParamErr
Net Libary
Net Library Functions

netErrSocketNotOpen
netErrSocketClosedByRemote
netErrOutOfCmdBlocks

Sockets Equivalent

int getpeername (int s, struct sockaddr *name, int *namelen);
int getsockname (int s, struct sockaddr *name, int *namelen);

Comments
This call is mainly useful for stream-based sockets. It allows the
caller to find out what address was bound to a connected socket and
the address of the remote host that it’s connected to.

In Palm OS version 3.0 and higher, if you pass a raw socket to this
function, it returns the instance number and creator of the interface
to which the socket is bound.

See Also
NetLibSocketBind, NetLibSocketConnect,
NetLibSocketAccept

NetLibSocketBind

Purpose
Assign a local address to a socket.

Declared In
NetMgr.h

Prototype
Int16 NetLibSocketBind (UInt16 libRefnum,
NetSocketRef socket,
NetSocketAddrType *sockAddrP, Int16 addrLen,
Int32 timeout, Err *errP)

Parameters
- > libRefNum Reference number of the net library.
- > socket Descriptor for the open socket.
- > sockAddrP Pointer to the address to give to the socket. This
can be a NetSocketAddrINType or a
NetSocketAddrRawType.
- > addrLen Length of address in *sockAddrP.
Net Library
Net Library Functions

- timeout Maximum timeout in system ticks; -1 means wait forever.

<- errP Contains an error code if the return value is -1.

Result Returns 0 upon success and -1 if an error occurred. If an error occurred, errP contains one of the following values:

0 No error.
netErrTimeout Call timed out.
netErrNotOpen The referenced net library has not been opened yet.

Sockets Equivalent int bind (int socket, const void *sockAddrP, int addrLen);

Comments Applications that want to wait for an incoming connection request from a remote host must call this function. After calling NetLibSocketBind, applications can call NetLibSocketListen and then NetLibSocketAccept to make the socket ready to accept connection requests.

Compatibility Raw sockets are only supported in Palm OS version 3.0 and higher. See NetLibSocketOpen for instructions on how to bind raw sockets.

See Also NetLibSocketConnect, NetLibSocketListen, NetLibSocketAccept
NetLibSocketClose

Purpose
Close a socket.

Declared In
NetMgr.h

Prototype
Int16 NetLibSocketClose (UInt16 libRefnum, NetSocketRef socket, Int32 timeout, Err *errP)

Parameters
- libRefNum Reference number of the net library.
- socket Descriptor for the open socket.
- timeout Maximum timeout in system ticks; -1 means wait forever.
- errP Contains an error code if the return value is -1.

Result
Returns 0 upon success and -1 if an error occurred. If an error occurred, errP contains one of the following values:

0 No error.
netErrTimeout Call timed out.
netErrNotOpen The referenced net library has not been opened yet.
netErrParamErr
netErrSocketNotOpen
netErrOutOfCmdBlocks

Sockets Equivalent
int close (int socket);

Comments
Closes down a socket and frees all memory associated with it.

See Also
NetLibSocketOpen, NetLibSocketShutdown
**NetLibrary**

**Net Library Functions**

---

**NetLibSocketConnect**

**Purpose**
Assign a destination address to a socket and initiate three-way handshake if it’s stream based.

**Declared In**
NetMgr.h

**Prototype**
```
Int16 NetLibSocketConnect (UInt16 libRefnum, NetSocketRef socket, NetSocketAddrType *sockAddrP, Int16 addrLen, Int32 timeout, Err *errP)
```

**Parameters**
- `-> libRefNum` Reference number of the net library.
- `-> socket` Descriptor for the open socket.
- `-> sockAddrP` Pointer to address to connect to.
- `-> addrLen` Length of address in `*sockAddrP`.
- `-> timeout` Maximum timeout in system ticks; -1 means wait forever.
- `<- errP` Contains an error code if the return value is -1.

**Result**
Returns 0 upon success and -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

- 0 No error.
- netErrTimeout Call timed out.
- netErrNotOpen The referenced net library has not been opened yet.
- netErrParamErr
- netErrSocketNotOpen
- netErrSocketBusy
- netErrNoInterfaces Incorrect setup.
- netErrPortInUse
- netErrQuietTimeNotElapsed
nullErrInternal
nullErrSocketAlreadyConnected
nullErrSocketClosedByRemote
nullErrTooManyTCPConnections
nullErrWouldBlock
nullErrWrongSocketType
nullErrOutOfCmdBlocks

**Sockets Equivalent**

```c
int connect (int socket, const void *sockAddrP, int addrLen);
```

**See Also**  
[NetLibSocketBind](#), [NetUTCPOpen](#)

---

**NetLibSocketListen**

**Purpose**  
Put a stream-based socket into passive listen mode.

**Declared In**  
NetMgr.h

**Prototype**  
```c
Int16 NetLibSocketListen (UInt16 libRefnum,
NetSocketRef socket, UInt16 queueLen,
Int32 timeout, Err *errP)
```

**Parameters**

- `libRefNum`  
  Reference number of the net library.

- `socket`  
  Descriptor for the open socket.

- `queueLen`  
  Maximum number of pending connections allowed.

- `timeout`  
  Maximum timeout in system ticks; -1 means wait forever.

- `errP`  
  Contains an error code if the return value is -1.

**Result**  
Returns 0 upon success and -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

- 0  
  No error.
netErrTimeout    Call timed out.
netErrNotOpen    The referenced net library has not been opened yet.
netErrParamErr
netErrOutOfResources
netErrSocketNotOpen
netErrSocketBusy
netErrNoInterfaces   Incorrect setup.
netErrPortInUse
netErrInternal
netErrSocketAlreadyConnected
netErrSocketClosedByRemote
netErrWrongSocketType
netErrQuietTimeNotElapsed
netErrOutOfCmdBlocks

**Sockets Equivalent**

```c
int listen (int socket, int queueLen);
```

**Comments**

Sets the maximum allowable length of the queue for pending connections. This call is only applicable to stream-based (TCP/IP) sockets.

After a socket is created and bound to a local address using [NetLibSocketBind](#), a server application can call NetLibSocketListen and then [NetLibSocketAccept](#) to accept connections from remote clients.

The `queueLen` is currently quietly limited to 1 (higher values are ignored).

**See Also**

[NetLibSocketBind](#), [NetLibSocketAccept](#)
**NetLibSocketOpen**

**Purpose**
Open a new socket.

**Declared In**
NetMgr.h

**Prototype**
```c
NetSocketRef NetLibSocketOpen (UInt16 libRefnum, NetSocketAddrEnum domain, NetSocketTypeEnum type, Int16 protocol, Int32 timeout, Err *errP)
```

**Parameters**
- `libRefNum`: Reference number of the net library.
- `domain`: Address domain. See `NetSocketAddrEnum`.
- `type`: Desired type of connection. See `NetSocketTypeEnum`.
- `protocol`: Protocol to use. This parameter is currently ignored.
  - For raw sockets in the `netSocketAddrINET` domain, specify one of the following:
    - `netSocketProtoIPTCP`
    - `netSocketProtoIPUDP`
    - `netSocketProtoIPRAW`
  - For all other socket types or for raw sockets in the raw domain, this parameter is ignored.
- `timeout`: Maximum timeout in system ticks; -1 means wait forever.
- `errP`: Contains an error code if the return value is -1.

**Result**
Returns the `NetSocketRef` of the opened socket or -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

- `0`: No error.
- `netErrTimeout`: Call timed out.
- `netErrNotOpen`: The referenced net library has not been opened yet.
Net Library
Net Library Functions

netErrParamErr
netErrNoMoreSockets
netErrOutOfCmdBlocks
netErrOutOfMemory

Sockets Equivalent

int socket (int domain, int type, int protocol);

Comments
Allocates memory for a new socket and opens it.

Raw sockets are supported in Palm OS version 3.0 and higher. Two types of raw sockets are supported:

- Raw sockets in the netSocketAddrINET domain
  In this case, you must bind the socket to an IP address using NetLibSocketBind, passing a NetSocketAddrINType structure for the socket address. The port field is ignored.
  
  For applications that use raw sockets in the INET domain, the net library checks the destination IP address of all incoming packets to see if it matches any of those raw sockets. If it does, the packet is enqueued directly into the matching socket and is not passed to the protocol stack.
  
  When an application sends data through raw sockets in the IP domain, the net library packages the data into a packet and passes it directly to the interface’s send routine. You are responsible for forming the entire IP header, including any necessary checksums, source and destination IP address, and so on.

- Raw sockets in the netSocketAddrRaw domain with no protocol
  In this case, you must bind the socket to an interface using NetLibSocketBind, passing a NetSocketAddrRawType structure for the socket address. The instance and creator specify which interface the caller wants to receive raw packets from.
  
  When an interface is bound to a raw socket with no protocol, the net library places that interface into raw mode. In raw
mode, the interface passes all incoming packets, no matter what the link layer protocol, to its raw receive function.

When an application sends data through a raw socket with no protocol, the net library packages the data into a packet and passes it directly to the interface’s send routine.

The interface remains in raw mode until the raw socket is closed.

**Compatibility**

Raw sockets supported only in Palm OS version 3.0 and higher.

**See Also**

NetLibSocketClose, NetUTCPOpen

### NetLibSocketOptionGet

**Purpose**

Retrieves the current value of a socket option.

**Declared In**

NetMgr.h

**Prototype**

```c
Int16 NetLibSocketOptionGet (UInt16 libRefnum, NetSocketRef socket, UInt16 level, UInt16 option, void *optValueP, UInt16 *optValueLenP, Int32 timeout, Err *errP)
```

**Parameters**

- `libRefNum` Reference number of the net library.
- `socket` Descriptor for the open socket.
- `level` Level of the option, one of the NetSocketOptLevelEnum constants. See NetLibSocketOptionSet.
- `option` One of the NetSocketOptEnum constants. See NetLibSocketOptionSet.
- `optValueP` Pointer to variable holding new value of option.
- `optValueLenP` Size of variable pointed to by optValueP on entry. Actual size of return value on exit.
timeout Maximum timeout in system ticks; -1 means wait forever.
errP Contains an error code if the return value is -1.

Result
Returns 0 upon success and -1 if an error occurred. If an error occurred, errP contains one of the following values:
- 0 No error.
- netErrTimeout Call timed out.
- netErrNotOpen The referenced net library has not been opened yet.
- netErrParamErr
- netErrSocketNotOpen
- netErrUnimplemented
- netErrWrongSocketType
- netErrInvalidSettingSize

Sockets Equivalent
int getsockopt (int socket, int level, int option, const void *optValueP, int *optValueLenP);

Comments
Returns the current value of a socket option. The caller passes a pointer to a variable to hold the returned value (in optValueP) and the size of this variable (in *optValueLenP). On exit, *optValueLenP is updated with the actual size of the return value.

For all of the fixed size options (every option except netSocketOptIPOptions), *optValueLenP is unmodified on exit and this call does its best to return the value in the caller’s desired type size.

For compatibility with existing Internet applications, this call is quite flexible on the *optValueLenP parameter. If the desired type for an option is FLAG, this call supports an *optValueLenP of 1, 2, or 4. If the desired type for an option is int, it supports an *optValueLenP of 2 or 4.

See NetLibSocketOptionSet for a list of available options.

See Also NetLibSocketOptionSet
**NetLibSocketOptionSet**

**Purpose**  Set a socket option.

**Declared In**  NetMgr.h

**Prototype**  
```c
Int16 NetLibSocketOptionSet (UInt16 libRefnum, NetSocketRef socket, UInt16 level, UInt16 option, void *optValueP, UInt16 optValueLen, Int32 timeout, Err *errP)
```

**Parameters**
- `libRefNum`  Reference number of the net library.
- `socket`  Descriptor for the open socket.
- `level`  Level of the option, one of the NetSocketOptLevelEnum constants. See the comments section.
- `option`  One of the NetSocketOptEnum constants. See the comments section.
- `optValueP`  Pointer to the variable holding the new value of the option.
- `optValueLen`  Size of variable pointed to by `optValueP`.
- `timeout`  Maximum timeout in system ticks; -1 means wait forever.
- `errP`  Contains an error code if the return value is -1.

**Result**  Returns 0 upon success and -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

- 0  No error.
- netErrTimeout  Call timed out.
- netErrNotOpen  The referenced net library has not been opened yet.
- netErrParamErr
- netErrSocketNotOpen
- netErrUnimplemented

---

*Palm OS Programmer’s API Reference*  1497
**Sockets Equivalent**

```c
int setsockopt (int socketRef, int level, int option, const void *optValueP, int optValueLen);
```

**Comments**

Sets various options associated with a socket. The caller passes a pointer to the new option value in `optValueP` and the size of the option in `optValueLen`.

Table 61.4 lists the available options.

- The Level column specifies the option level, which is one of the `netSocketOptLevelXXX` constants.
- The Option column lists the option, which is one of the `netSocketOptXXX` constants.
- The G/S column lists whether this option can be fetched with the `NetLibSocketOptionGet` call (G) and/or set (S) with this call.
- The type column lists the data type of the option.
- The I column specifies whether or not this option is currently implemented.

**Table 61.4 Net Library Socket Options**

<table>
<thead>
<tr>
<th>netSocket OptLevel</th>
<th>netSocketOpt...</th>
<th>G/S</th>
<th>Type</th>
<th>I</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP</td>
<td>IPOptions</td>
<td>GS</td>
<td>UInt8[]</td>
<td>N</td>
<td>Options in IP Header</td>
</tr>
<tr>
<td>TCP</td>
<td>TCPNoDelay</td>
<td>GS</td>
<td>FLAG</td>
<td>Y</td>
<td>Don’t delay send to coalesce packets</td>
</tr>
<tr>
<td>TCP</td>
<td>TCPMaxSeg</td>
<td>G</td>
<td>int</td>
<td>Y</td>
<td>Get TCP maximum segment size</td>
</tr>
<tr>
<td>Socket</td>
<td>SockDebug</td>
<td>GS</td>
<td>FLAG</td>
<td>N</td>
<td>Turn on recording of debug info</td>
</tr>
<tr>
<td>Socket</td>
<td>SockAcceptConn</td>
<td>G</td>
<td>FLAG</td>
<td>N</td>
<td>Socket has had listen</td>
</tr>
</tbody>
</table>
Table 61.4 Net Library Socket Options (continued)

<table>
<thead>
<tr>
<th>netSocket OptLevel...</th>
<th>netSocketOpt...</th>
<th>G/S</th>
<th>Type</th>
<th>I</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket</td>
<td>SockReuseAddr</td>
<td>GS</td>
<td>FLAG</td>
<td>N</td>
<td>Allow local address reuse</td>
</tr>
<tr>
<td>Socket</td>
<td>SockKeepAlive</td>
<td>GS</td>
<td>FLAG</td>
<td>Y</td>
<td>Keep connections alive</td>
</tr>
<tr>
<td>Socket</td>
<td>SockDontRoute</td>
<td>GS</td>
<td>FLAG</td>
<td>N</td>
<td>Just use interface addresses</td>
</tr>
<tr>
<td>Socket</td>
<td>SockBroadcast</td>
<td>GS</td>
<td>FLAG</td>
<td>N</td>
<td>Permit sending of broadcast messages</td>
</tr>
<tr>
<td>Socket</td>
<td>SockUseLoopback</td>
<td>GS</td>
<td>FLAG</td>
<td>N</td>
<td>Bypass hardware when possible</td>
</tr>
<tr>
<td>Socket</td>
<td>SockLinger</td>
<td>GS</td>
<td>NetSocke tLingerType</td>
<td>Y</td>
<td>Linger on close if data present NetSocketLingerType is a structure with two fields: onOff (true or false) and time (linger time in seconds).</td>
</tr>
<tr>
<td>Socket</td>
<td>SockOOBInLine</td>
<td>GS</td>
<td>FLAG</td>
<td>N</td>
<td>Leave received OOB data in-line</td>
</tr>
<tr>
<td>Socket</td>
<td>SockSndBufSize</td>
<td>GS</td>
<td>int</td>
<td>N</td>
<td>Send buffer size</td>
</tr>
<tr>
<td>Socket</td>
<td>SockRcvBufSize</td>
<td>GS</td>
<td>int</td>
<td>N</td>
<td>Receive buffer size</td>
</tr>
<tr>
<td>Socket</td>
<td>SockSndLowWater</td>
<td>GS</td>
<td>int</td>
<td>N</td>
<td>Send low-water mark</td>
</tr>
<tr>
<td>Socket</td>
<td>SockRcvLowWater</td>
<td>GS</td>
<td>int</td>
<td>N</td>
<td>Receive low-water mark</td>
</tr>
<tr>
<td>Socket</td>
<td>SockSndTimeout</td>
<td>GS</td>
<td>int</td>
<td>N</td>
<td>Send timeout</td>
</tr>
<tr>
<td>Socket</td>
<td>SockRcvTimeout</td>
<td>GS</td>
<td>int</td>
<td>N</td>
<td>Receive timeout</td>
</tr>
<tr>
<td>Socket</td>
<td>SockErrorStatus</td>
<td>G</td>
<td>int</td>
<td>Y</td>
<td>Get error status and clear</td>
</tr>
<tr>
<td>Socket</td>
<td>SockSocketType</td>
<td>G</td>
<td>int</td>
<td>Y</td>
<td>Get socket type</td>
</tr>
</tbody>
</table>
For compatibility with existing Internet applications, this call is quite flexible on the \textit{optValueLen} parameter. If the desired type for an option is \texttt{FLAG}, this call accepts an \textit{optValueLen} of 1, 2, or 4. If the desired type for an option is \texttt{int}, it accepts an \textit{optValueLen} of 2 or 4.
Except for the `netSocketOptSockNonBlocking` option, all options listed above have equivalents in the sockets API. The `netSocketOptSockNonBlocking` option was added to this call in the net library in order to implement the functionality of the UNIX `fcntl()` control call, which can be used to turn nonblocking mode on and off for sockets.

**See Also**  
`NetLibSocketOptionGet`  
`NetLibSocketShutdown`  

**NetLibSocketShutdown**

**Purpose**  
Shut down a socket in one or both directions.

**Declared In**  
`NetMgr.h`

**Prototype**  
```
Int16 NetLibSocketShutdown (UInt16 libRefnum,  
 NetSocketRef socket, Int16 direction,  
 Int32 timeout, Err *errP)
```  

**Parameters**

- `libRefNum`  
  Reference number of the net library.

- `socket`  
  Descriptor for the open socket.

- `direction`  
  Direction to shut down. One of the `NetSocketDirEnum` constants. Specifically:
  - `netSocketDirInput`
  - `netSocketDirOutput`
  - `netSocketDirBoth`

- `timeout`  
  Maximum timeout in system ticks; -1 means wait forever.

- `errP`  
  Contains an error code if the return value is -1.

**Result**

Returns 0 upon success and -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

- 0  
  No error.

- `netErrTimeout`  
  Call timed out.

- `netErrNotOpen`  
  The referenced net library has not been opened yet.
**netErrParamErr**
**netErrSocketNotOpen**
**netErrNoMultiPktAddr**
**netErrOutOfCmdBlocks**

**Sockets Equivalent**
`int shutdown (int socket, int direction);`

**Comments**
Shuts down communication in one or both directions on a socket.
If direction is `netSocketDirInput`, the socket is marked as down in the receive direction and further read operations from it return a `netErrSocketInputShutdown` error.

---

**NetLibTracePrintf**

**Purpose**
Store debugging information in the net library’s trace buffer.

**Declared In**
`NetMgr.h`

**Prototype**
```c
Err NetLibTracePrintf (UInt16 libRefNum, const Char *formatStr, ...)
```

**Parameters**
- `libRefNum` Reference number of the net library.
- `formatStr` A `printf` style format string.
- `...` Arguments to the format string.

**Result**
Returns 0 upon success or `netErrNotOpen` if the net library has not been opened.

**Sockets Equivalent**
None

**Comments**
This call is a convenient debugging tool for developing Internet applications. It stores a message into the net library’s trace buffer, which can later be dumped using the `NetLibMaster` call. The net library’s trace buffer is used to store run-time errors that the net
Net Library
Net Library Functions

library encounters as well as errors and messages from network interfaces and from applications that use this call.

The `formatStr` parameter is a `printf` style format string which supports the following format specifiers:

- `%d`, `%i`, `%u`, `%x`, `%s`, `%c`

but it does **not** support field widths, leading 0’s etc.

Note that the `netTracingAppMsgs` bit of the `netSettingTraceBits` setting must be set using the call `NetLibSettingSet(...netSettingTraceBits...)`. Otherwise, this routine will do nothing.

**See Also**  
`NetLibTracePutS`, `NetLibMaster`, `NetLibSettingSet`

### NetLibTracePutS

**Purpose** Store debugging information in the net library’s trace buffer.

**Declared In** `NetMgr.h`

**Prototype**

```c
Err NetLibTracePutS (UInt16 libRefNum, 
                    Char *strP)
```

**Parameters**

- `libRefNum` Reference number of the net library.
- `strP` String to store in the trace buffer.

**Result** Returns 0 upon success or `netErrNotOpen` if the net library has not been opened.

**Sockets Equivalent** None

**Comments** This call is a convenient debugging tool for developing Internet applications. It will store a message into the net library’s trace buffer which can later be dumped using the `NetLibMaster` call. The net library’s trace buffer is used to store run-time errors that the net library encounters as well as errors and messages from network interfaces and from applications that use this call.
Note the netTracingAppMsgs bit of the netSettingTraceBits setting must be set using the NetLibSettingSet(...netSettingTraceBits...) call or this routine will do nothing.

**NetNToHL**

**Purpose**  Macro that converts a 32-bit value from network to host byte order.

**Declared In**  NetBitUtils.h

**Prototype**  NetNToHL (x)

**Parameters**  -> x  32-bit value to convert.

**Result**  Returns x in host byte order.

**Errors**  none

**Sockets Equivalent**  ntohl()

**See Also**  NetNToHS, NetHToNL, NetHToNS

**NetNToHS**

**Purpose**  Macro that converts a 16-bit value from network to host byte order.

**Declared In**  NetBitUtils.h

**Prototype**  NetNToHS (x)

**Parameters**  -> x  16-bit value to convert.

**Result**  Returns x in host byte order.
Errors  None

Sockets Equivalent  ntohs()  

See Also  NetHToNL, NetNToHL, NetHToNS
Network Utilities

This chapter describes network utilities provided in the module NetSocket.c. These utilities are convenience functions that you can use in place of net library functions in applications that use the net library. You can find NetSocket.c in the folder Libraries\Net\Src. (On Palm OS® 3.5, NetSocket.c is in the folder CodeWarrior Libraries\Comms\NetSocket\Src.)

The include file for the functions described in this chapter is <unix/sys_socket.h>. This header file is not included by any other Palm™ header file; you must explicitly include it in your code.

For more information on NetSocket.c and sys_socket.h, see the chapter “Network Communication” in the Palm OS Programmer’s Companion, vol. II, Communications.

Network Utility Functions

NetUReadN

Purpose
Reads a specified number of bytes from a socket.

Declared In  unix/sys_socket.h

Prototype
Int32 NetUReadN (NetSocketRef fd, UInt8* bufP, UInt32 numBytes)

Parameters
- fd  Descriptor for the open socket.
- bufP  Pointer to buffer to hold received data.
- numBytes  Number of bytes to read.

Result
Returns the number of bytes actually read. If the return value is less than 0, an error occurred.
**Network Utilities**  
*Network Utility Functions*

**Comments**  
This function repeatedly calls NetLibReceive until numBytes have been read or until NetLibReceive returns an error.

**See Also**  
NetUWriteN

**NetUTCPOpen**

**Purpose**  
Opens a TCP (streams-based) socket and connects it to a server.

**Declared In**  
unix/sys_socket.h

**Prototype**  
NetSocketRef NetUTCPOpen (Char* hostName, Char* serviceName, Int16 port)

**Parameters**

- **-> hostName**  
Remote host, given either by name or by dotted decimal address.

- **-> serviceName**  
The name of a network service. Possible services are “echo”, “discard”, “daytime”, “qotd”, “chargen”, “ftp-data”, “ftp”, “telnet”, “smtp”, “time”, “name”, “finger”, “pop2”, “pop3”, “nntp”, “imap2”. The value of this parameter is ignored if the port parameter is greater than zero.

- **-> port**  
The number of the port to connect to on the remote host. Set port to zero to use serviceName instead.

**Result**  
Returns the socket descriptor of the socket that was connected, or -1 if an error occurred.

**Comments**  
If serviceName is given and port is less than or equal to zero, this function looks up the port number for that service on the remote host and uses it for the connection.

This function is the equivalent of calling NetLibSocketOpen and NetLibSocketConnect (or socket and connect).
NOTE: This function does not return specific reasons for failure if there is a failure. This function is not production-quality code. It is provided as a quick and dirty way of creating a connection and as sample code that can be used as a reference.

NetUWriteN

Purpose  Writes the specified number of bytes to a socket.

Declared In  unix/sys_socket.h

Prototype  Int32 NetUWriteN (NetSocketRef fd, UInt8* bufP,(UInt32 numBytes)

Parameters  -> fd  Descriptor for the open socket.
-   -> bufP  Pointer to buffer to write.
-   -> numBytes  Number of bytes to write.

Result  Returns the number of bytes actually sent. If the return value is less than 0, an error occurred.

Comments  This function repeatedly calls NetLibSend until numBytes have been written or until NetLibSend returns an error.

See Also  NetUReadN
Script Plugin

This chapter describes the login script plugin support. You write a plugin to add to the list of available login script commands in the Network preferences panel. This chapter covers:

- Script Plugin Data Types
- Script Plugin Constants

The header file `ScriptPlugin.h` declares the API described in this chapter.

For more information on the script plugin, see the section “Extending the Network Login Script Support” on page 185 in the “Network Communication” chapter of the *Palm OS Programmer's Companion*, vol. II, Communications.

Script Plugin Data Types

**PluginCallbackProcType**

The `PluginCallbackProcType` defines the `procP` field in `PluginExecCmdType`.

```c
typedef struct {
    ScriptPluginSelectorProcPtr selectorProcP;
} PluginCallbackProcType,
*PluginCallbackProcPtr;
```

**Field Descriptions**


---

Palm OS Programmer’s API Reference 1511
**PluginCmdPtr**

The `PluginCmdPtr` type defines a pointer to a `PluginCmdType` structure.

```
typedef PluginCmdType * PluginCmdPtr;
```

**PluginCmdType**

The `PluginCmdType` structure specifies the name of a command.

```
typedef struct {
    Char    commandName[pluginMaxCmdNameLen + 1];
    Boolean hasTxtStringArg;
    UInt8   reserved;
} PluginCmdType;
```

**Field Descriptions**

- **commandName**
  - The name of the command. This string appears in the pull-down list in the Network preferences panel’s script view. The pull-down list contains all available commands from all plugins. Make sure that your command name is unique and as short as possible.

- **hasTxtStringArg**
  - `true` if the command takes an argument. In this case when the user selects this command, the Network preferences panel displays a field next to the command name where the user should enter the argument. This argument is passed in the `txtStringArg` field in `PluginExecCmdType` when the command is to be executed.

- **reserved**
  - Reserved for future use.
PluginExecCmdType

The PluginExecCmdType structure defines the parameter block for the scptLaunchCmdExecuteCmd launch code. This structure specifies which command is to be executed and provides any necessary arguments for the command. Your plugin should respond by executing the command.

```
typedef struct {
    Char    commandName[pluginMaxCmdNameLen + 1];
    Char    txtStringArg [pluginMaxLenTxtStringArg + 1];
    PluginCallbackProcPtr procP;
    void *  handle;
} PluginExecCmdType, *PluginExecCmdPtr;
```

**Field Descriptions**

**commandName**  The command’s name. This is the string that appears in the pull-down list in the script view of the Network preferences panel.

**txtStringArg**  If the command takes an argument, this field provides the argument as a string. A NULL value means either that the user did not provide a value, or that you didn’t specify that the command takes an argument.

**procP**  A pointer to a PluginCallbackProcType structure, which identifies the network interface function that the plugin can use to execute the command.

**handle**  Handle to information specific to a particular connection. You must pass this value when you call the function pointed to by procP.
**PluginInfoPtr**

The `PluginInfoPtr` type defines a pointer to a `PluginInfoType` structure.

```c
typedef PluginInfoType * PluginInfoPtr;
```

**PluginInfoType**

The `PluginInfoType` structure is the parameter block for the `scptLaunchCmdListCmds` launch code. When your plugin receives the launch code, the `PluginInfoType` structure is empty. Your plugin should fill in the `PluginInfoType` and return it. The system uses the information returned to construct the pull-down list of available script commands and build a table of which plugin will execute which script command.

```c
typedef struct {
    Char  pluginName[pluginMaxModuleNameLen + 1];
    UInt16  numOfCommands;
    PluginCmdType  command[pluginMaxNumOfCmds];
} PluginInfoType;
```

**Field Descriptions**

- **pluginName**  
  A name that the system can use to identify your plugin. This is typically the same name you give the PRC file.

- **numOfCommands**  
  The number of commands that your plugin defines. The maximum allowed is `pluginMaxNumOfCmds`.

- **command**  
  An array of `PluginCmdType` structures that provide information about the commands that your plugin defines.
ScriptPluginLaunchCodesEnum

The ScriptPluginLaunchCodesEnum defines the launch codes for the script plugin. Your script plugin’s PilotMain function should respond to the launch codes defined in this enum.

typedef enum {
    scptLaunchCmdDoNothing = sysAppLaunchCmdCustomBase,
    scptLaunchCmdListCmds,
    scptLaunchCmdExecuteCmd
} ScriptPluginLaunchCodesEnum;

Value Descriptions

scptLaunchCmdDoNothing This launch code is a no-op supplied only to provide a beginning value for the script plugin launch codes. It is not necessary to respond to this launch code.

scptLaunchCmdListCmds Provide information about the commands that your plugin executes. See PluginInfoType.

scptLaunchCmdExecuteCmd Execute the specified command.

This launch code is received when the system is executing a user’s login script during a network connection attempt. Your plugin should respond by executing the command provided in the PluginExecCmdType parameter block.
Script Plugin Constants

Command Constants

The following constants identify the available commands that the network interface can perform for you. These commands are building blocks that you use to create your own script commands. To perform one of these tasks, pass the constant value as an argument to the network interface’s callback function ([ScriptPluginSelectorProc](#)).

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pluginNetLibDoNothing</td>
<td>0</td>
<td>For debugging purposes.</td>
</tr>
<tr>
<td>pluginNetLibReadBytes</td>
<td>1</td>
<td>Read the specified number of bytes from the open connection.</td>
</tr>
<tr>
<td>pluginNetLibWriteBytes</td>
<td>2</td>
<td>Write the specified number of bytes to the open connection.</td>
</tr>
<tr>
<td>pluginNetLibGetUserName</td>
<td>3</td>
<td>Get the user name from the network service profile.</td>
</tr>
<tr>
<td>pluginNetLibGetUserPwd</td>
<td>4</td>
<td>Get the user’s password from the network service profile.</td>
</tr>
<tr>
<td>pluginNetLibCheckCancelStatus</td>
<td>5</td>
<td>Check to see if the user canceled the connection.</td>
</tr>
<tr>
<td>pluginNetLibPromptUser</td>
<td>6</td>
<td>Prompt the user for input.</td>
</tr>
<tr>
<td>pluginNetLibConnLog</td>
<td>7</td>
<td>Write a string to the network service’s connection log.</td>
</tr>
<tr>
<td>pluginNetLibCallUIProc</td>
<td>8</td>
<td>Have the network interface call a function in your plugin that displays UI.</td>
</tr>
</tbody>
</table>

Palm OS Programmer’s API Reference
## Script Plugin

### Script Plugin Constants

The following table lists constants that control the size of strings in your plugin and the size of the plugin itself.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pluginNetLibGetSerLibRefNum</td>
<td>9</td>
<td>Obtain the serial library’s reference number. You need the reference number to perform any serial library commands, which you might need to perform more complex work with the connection port.</td>
</tr>
<tr>
<td>pluginMaxCmdNameLen</td>
<td>15</td>
<td>The maximum length for the command’s name, not including the terminating null character. This is the string displayed to the user in the pull-down menu.</td>
</tr>
<tr>
<td>pluginMaxModuleNameLen</td>
<td>15</td>
<td>The maximum length for the plugin’s name (not including the terminating null character), which is typically the name of the PRC file as well.</td>
</tr>
<tr>
<td>pluginMaxNumOfCmds</td>
<td>10</td>
<td>The maximum number of commands that your plugin can define.</td>
</tr>
<tr>
<td>pluginMaxLenTxtStringArg</td>
<td>63</td>
<td>The maximum length of the argument that each command can take, not including the terminating null character.</td>
</tr>
</tbody>
</table>
Script Plugin Functions

**ScriptPluginSelectorProc**

**Purpose**
A function provided by the network interface for the purpose of performing script commands.

**Declared In**
ScriptPlugin.h

**Prototype**
Err (*ScriptPluginSelectorProcPtr) (void *handle, UInt16 command, void *dataBufferP, UInt16 *sizeP, UInt16 *dataTimeoutP, void *procAddrP);

**Parameters**
- **handle**
  Handle to information specific to a particular connection.

- **command**
  The command to be executed. See “Command Constants” for a list of possible values. The rest of the parameters to this callback function are interpreted differently based on the value of the command parameter. See the table in the “Comments” section for specifics.

- **dataBufferP**
  A pointer to arguments to pass to the command or a pointer to data returned by the command. See the “Comments” section below.

- **sizeP**
  The size of dataBufferP.

- **dataTimeoutP**
  Number of seconds to wait for the command to execute. 0 means wait forever. Applies only to commands that request information from the network.
- `procAddrP`  
  Pointer to a user interface callback function that the network interface should call to complete the function. Used only by `pluginNetLibCallUIProc`. This function should take one argument of the same type that you pass to `dataBufferP` and should return `void`.

**Result**  
Returns 0 upon success, or an error condition upon failure. If an error condition is returned, your plugin should stop processing and return the error condition from its `PilotMain`.

**Comments**  
When your plugin receives the `scptLaunchCmdExecuteCmd` launch code, the parameter block contains the command’s name, its text string argument (if any), and a pointer to the network interface’s callback function. You should use this callback function any time you need to communicate with the network library, the user, or the host computer during execution of your command.

The callback function takes as arguments the handle to information about this connection (which is also passed in the launch code’s parameter block), and the command that the service should execute. The rest of the parameters are interpreted differently based on what the value the command argument is. See the table below.

<table>
<thead>
<tr>
<th><code>pluginNetLib</code></th>
<th><code>dataBufferP</code></th>
<th><code>sizeP</code></th>
<th><code>dateTimeOutP</code></th>
<th><code>procAddrP</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>DoNothing</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ReadBytes</td>
<td>On return, contains the bytes that were read.</td>
<td>On input, contains the number of bytes to read.</td>
<td>Number of seconds to wait before timing out the operation.</td>
<td>N/A</td>
</tr>
</tbody>
</table>

On return, contains the number of bytes actually read.
<table>
<thead>
<tr>
<th>PluginNetLib</th>
<th>dataBufferP</th>
<th>sizeP</th>
<th>deltaTimeOutP</th>
<th>procAddrP</th>
</tr>
</thead>
<tbody>
<tr>
<td>WriteBytes</td>
<td>On input, contains the data to send.</td>
<td>On input, contains the number of bytes to send.</td>
<td>Number of seconds to wait for a response before canceling.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>On return, contains the number of bytes actually sent.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UserName</td>
<td>On return, contains the user’s name</td>
<td>On return, contains the size of the string pointed to by dataBufferP.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>UserPwd</td>
<td>On return, contains the user’s password.</td>
<td>On return, contains the size of the string pointed to by dataBufferP.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>CheckCancel</td>
<td>On return, the Boolean value true if the user canceled the command, false otherwise.</td>
<td>Size of Boolean.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Status</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PromptUser</td>
<td>On input, the prompt to display.</td>
<td>On input and on return, the size of the string pointed to by dataBufferP.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Script Plugin

#### Script Plugin Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnLog</td>
<td>The string that should be written to the log.</td>
</tr>
<tr>
<td>CallUIProc</td>
<td>A pointer to a structure to pass to your callback function as a parameter. This structure should contain a handle to the form to be displayed, plus any other necessary information.</td>
</tr>
<tr>
<td>GetSerLib RefNum</td>
<td>On return, contains the serial library’s reference number.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pluginNetLib</td>
<td>N/A</td>
</tr>
<tr>
<td>dataBufferP</td>
<td>N/A</td>
</tr>
<tr>
<td>sizeP</td>
<td>N/A</td>
</tr>
<tr>
<td>dateTimeOutP</td>
<td>A pointer to a function in your plugin that displays the form.</td>
</tr>
<tr>
<td>procAddrP</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Virtual Drivers

This chapter provides reference material for the Serial Manager virtual device driver API:

- **Driver Data Structures**
- **Driver Constants**
- **Virtual Driver-Defined Functions**
- **Serial Manager Queue Functions**

The header files SerialVdrv.h and SerialDrvr.h declare the virtual driver API. For more information on writing device drivers for the Serial Manager, see section “Writing a Virtual Device Driver” on page 114 in the “Serial Communication” chapter of Palm OS Programmer’s Companion, vol. II, Communications.

**Driver Data Structures**

**DrvInfoType**

The DrvInfoType structure defines information about the serial hardware. It is passed to and filled in by the DrvEntryPointProcPtr for a virtual driver.

```c
typedef struct {
    UInt32 drvrID;
    UInt32 drvrVersion;
    UInt32 maxBaudRate;
    UInt32 handshakeThreshold;
    UInt32 portFlags;
    const Char *portDesc;
    DrvrIRQEnum irqType;
    UInt8 multipleEnumerations;
    UInt32 dbCreator;
} DrvInfoType;
```
Virtual Drivers
Driver Data Structures

Value Descriptions

drvrID 4-character creator type, such as 'u328'.
drvrVersion Version of code that works for this hardware. See Driver Version Constants.
maxBaudRate Maximum baud rate supported by this hardware.
handshakeThreshold Baud rate at which the use of hardware handshaking is necessary.
portFlags Bit flags denoting features of this hardware. The flags are described in Port Feature Constants.
portDesc Pointer to null-terminated string describing this hardware. This string appears in the Connection panel to describe the port to the user (only if the portCncMgrVisible bit in portFlags is set). Can be NULL if the driver contains a resource (of type 'tSTR' and id kPortDescStrID) that supplies this string.
irqType IRQ line being used for this hardware. For a virtual driver, specify drvrIRQNone.
multipleEnumerations The number of entries in the driver table required for this driver. If 0, the driver has a single entry.
dbCreator Creator ID of the database containing this driver.

Compatibility The multipleEnumerations and dbCreator fields are only defined if New Serial Manager Feature Set Version 2 is present.
**DrvrRcvQType**

The `DrvrRcvQType` structure defines the virtual driver receive buffer and function pointers to functions that access and save data to the buffer. A pointer to this structure is passed to the `VdrvOpenProcPtr` function. The `DrvrHWRcvQPtr` type defines a pointer to a `DrvrRcvQType` structure.

```c
typedef struct DrvrRcvQType {
    void *rcvQ;
    WriteByteProcPtr qWriteByte;
    WriteBlockProcPtr qWriteBlock;
    GetSizeProcPtr qGetSize;
    GetSpaceProcPtr qGetSpace;
    SignalCheckPtr qSignalCheck;
} DrvrRcvQType;

typedef DrvrRcvQType *DrvrHWRcvQPtr;
```

**Value Descriptions**

- **rcvQ**  
  Pointer to the receive buffer.

- **qWriteByte**  
  Function pointer to a function that the virtual driver can use to write one byte to the Serial Manager’s receive queue. See the `WriteByteProcPtr` function.

- **qWriteBlock**  
  Function pointer to a function that the virtual driver can use to write a block of bytes to the Serial Manager’s receive queue. See the `WriteBlockProcPtr` function.

- **qGetSize**  
  Function pointer to a function that the virtual driver can use to get the total size of the Serial Manager’s receive queue. See the `GetSizeProcPtr` function.
Virtual Drivers
Driver Data Structures

qGetSpace Function pointer to a function that the virtual
driver can use to get the available space in the
Serial Manager’s receive queue. See the
GetSpaceProcPtr function.

qSignalCheck Function pointer to a function that the virtual
driver can use to perform a signal check for the
Serial Manager’s receive queue. See the
SignalCheckPtr function.

Compatibility The qSignalCheck field is only defined if New Serial Manager
Feature Set Version 2 is present.

DrvrStatusEnum
The DrvrStatusEnum enumerated type specifies serial status bit
flags. Return these enumerated types from the
VdrvStatusProcPtr call.

typedef enum DrvrStatusEnum {
  drvrStatusCtsOn = 0x0001,
  drvrStatusRtsOn = 0x0002,
  drvrStatusDsrOn = 0x0004,
  drvrStatusTxFifoFull = 0x0008,
  drvrStatusTxFifoEmpty = 0x0010,
  drvrStatusBreakAsserted = 0x0020,
  drvrStatusDataReady = 0x0040,
  drvrStatusLineErr = 0x0080
} DrvrStatusEnum;

Value Descriptions

drvrStatusCtsOn Set if CTS line is active.
drvrStatusRtsOn Set if RTS line is active.
drvrStatusDsrOn Set if DSR is on.
drvrStatusTxFifoFull Set if transmit FIFO is full; cleared
  if FIFO has space.
drvrStatusTxFifoEmpty Set if transmit FIFO is empty.
SrmRcvQType

The SrmRcvQType structure defines the Serial Manager receive queue. This queue is passed as a parameter to the virtual driver.

typedef struct SrmRcvQType {
    UInt32 qStart;
    UInt32 qEnd;
    UInt32 qSize;
    UInt8 *qData;
    void *qPort;
} SrmRcvQType;

Field Descriptions

qStart The start of the queue.
qEnd The end of the queue.
qSize The size of the queue.
qData The data currently in the queue.
qPort A pointer to the current foreground port.

Compatibility

The SrmRcvQType structure was previously a private structure. It is declared publicly if New Serial Manager Feature Set Version 2 is present.

VdrvAPIType

The VdrvAPIType structure defines function pointers to the required virtual driver functions. When passed a pointer to this structure in the DrvEntryPointProcPtr function, that function must fill in the pointers to the virtual driver functions appropriately.
typedef struct {
    VdrvOpenProcPtr drvOpen;
    VdrvCloseProcPtr drvClose;
    VdrvControlProcPtr drvControl;
    VdrvStatusProcPtr drvStatus;
    VdrvReadProcPtr drvRead;
    VdrvWriteProcPtr drvWrite;
    VdrvOpenProcV4Ptr drvOpenV4;
    VdrvControlCustomProcPtr drvControlCustom;
} VdrvAPIType;

**Value Descriptions**

- **drvOpen** Pointer to the driver open function.
- **drvClose** Pointer to the driver close function.
- **drvControl** Pointer to the driver control function.
- **drvStatus** Pointer to the driver status function.
- **drvRead** Pointer to the driver read function.
- **drvWrite** Pointer to the driver write function.
- **drvOpenV4** Pointer to the driver open function for New Serial Manager Feature Set Version 2.
- **drvControlCustom** Pointer to the driver custom control function.

**Compatibility**

drvOpenV4 and drvControlCustom are declared if both New Serial Manager Feature Set Version 2 and 4.0 New Feature Set are present.

**VdrvConfigType**

The VdrvConfigType structure specifies parameters for opening a serial port. This structure is passed as a parameter to VdrvOpenProcV4Ptr.

```c
typedef struct VdrvConfigType {
    UInt32 baud;
    UInt32 drvrId;
} VdrvConfigType;
```
Virtual Drivers
Driver Data Structures

UInt32 function;
MemPtr drvrDataP;
UInt16 drvrDataSize;
UInt32 sysReserved1;
UInt32 sysReserved2;
} VdrvConfigType;

Field Descriptions

 baud           Baud rate at which to open the connection. Serial drivers that do not require baud rates ignore this field.

 drvrId          Creator ID of the application or library that is using the Serial Manager.

 function       The reason why the port was opened. Specify the creator ID of the application that is opening the port or one of the following values:

       serFncUndefined       Undefined function. This is the default value for this field.

       serFncPPPSession     The connection is to be used for the PPP interface.

       serFncSLIPSession    The connection is to be used for the SLIP session.

       serFncDebugger       The connection is to be used for a debugging session.

       serFncHotSync        The connection is to be used for a HotSync operation.

       serFncConsole        The connection is to the debugging console.

       serFncTelephony      The connection is to the telephony library.
The function field is used by protocols such as USB and Bluetooth that perform different setup tasks based on which type of application is using them. RS-232 drivers ignore this parameter.

- **drvrDataP**: Pointer to a driver-specific data block.
- **drvrDataSize**: The size of the data block pointed to by drvrDataP.
- **sysReserved1**: Reserved for future use.
- **sysReserved2**: Reserved for future use.

**Compatibility**

This structure is only defined if both New Serial Manager Feature Set Version 2 and 4.0 New Feature Set are present.

**VdrvCtlOpCodeEnum**

The `VdrvCtlOpCodeEnum` enumerated type specifies a serial control operation. You should handle each of these constants when passed for the `controlCode` parameter to the `VdrvControlProcPtr` call.

```c
typedef enum VdrvCtlOpCodeEnum {
    vdrvOpCodeNoOp = 0,
    vdrvOpCodeSetBaudRate = 0x1000,
    vdrvOpCodeSetSettingsFlags,
    vdrvOpCodeSetCtsTimeout,
    vdrvOpCodeClearErr,
    vdrvOpCodeSetSleepMode,
    vdrvOpCodeSetWakeupMode,
    vdrvOpCodeFIFOCount,
    vdrvOpCodeStartBreak,
    vdrvOpCodeStopBreak,
    vdrvOpCodeStartLoopback,
    vdrvOpCodeStopLoopback,
    vdrvOpCodeFlushTxXFIFO,
    vdrvOpCodeFlushRxXFIFO,
    vdrvOpCodeSendBufferedData,
    vdrvOpCodeRcvCheckIdle,
    vdrvOpCodeEmuSetBlockingHook,
} VdrvCtlOpCodeEnum;
```
vdrvOpCodeGetOptTransmitSize,
vdrvOpCodeGetMaxRcvBlockSize,
vdrvOpCodeNotifyBytesReadFromQ,
vdrvOpCodeSetDTRasserted,
vdrvOpCodeGetDTRasserted,
vdrvOpCodeWaitForConfiguration,
vdrvOpCodeGetUSBDeviceDescriptor,
vdrvOpCodeGetUSBCfgDescriptor,
vdrvOpCodeEnableIRDA,
vdrvOpCodeDisableIRDA,
vdrvOpCodeEnableUART,
vdrvOpCodeDisableUART,
vdrvOpCodeRxEnable,
vdrvOpCodeRxDisable,
vdrvOpCodeLineEnable,
vdrvOpCodeEnableUARTInterrups,
vdrvOpCodeDisableUARTInterrups,
vdrvOpCodeSetReceiveQueue,
vdrvOpCodeSaveState,
vdrvOpCodeRestoreState,
vdrvOpCodeSetYieldPortCallback,
vdrvOpCodeSetYieldPortRefCon,
vdrvOpCodeUserDef = 0x2000,
vdrvOpCodeSystem = 0x7000,
vdrvOpCodeCustom = 0x8000
} VdrvCtlOpCodeEnum;

Value Descriptions

vdvrOpCodeSetBaudRate           Sets the baud rate.
vdvrOpCodeSetSettingsFlags      Sets the data transmission options. The bit
                                flags are described in Serial Settings
                                Constants.
vdvrOpCodeSetCtsTimeout         Hardware handshake timeout.
vdvrOpCodeClearErr              Clears the hardware error state.
vdvrOpCodeSetSleepMode          Puts the port in sleep mode (not typically
                                used for virtual drivers).
Virtual Drivers
Driver Data Structures

vdvrOpCodeSetWakeupMode
Wakes up the port from sleep mode (not typically used for virtual drivers).

vdvrOpCodeFIFOCount
Returns the number of bytes currently in the FIFO (or best estimate).

vdvrOpCodeStartBreak
Sends a break character or enables the sending of break characters.

vdvrOpCodeStopBreak
Stops sending break characters.

vdvrOpCodeStartLoopback
Starts loopback mode (not typically used for virtual drivers).

vdvrOpCodeStopLoopback
Stops loopback mode (not typically used for virtual drivers).

vdrvOpCodeFlushTxFIFO
Flushes the contents of the transmit FIFO.

vdrvOpCodeFlushRxFIFO
Flushes the contents of the receive FIFO.

vdrvOpCodeSendBufferedData
Notifies virtual device to send any buffered data it has not emptied from its internal buffers.

vdrvOpCodeRcvCheckIdle
Called periodically to allow the virtual device time to check if there is data to be received. Because virtual devices execute in the same thread as applications, they can be prevented from handling notifications of received data.

vdrvOpCodeEmuSetBlockingHook
Special op code for the Simulator.

vdrvOpCodeGetOptTransmitSize
Returns the optimum buffer size for sending data or returns 0 to specify any buffer size is acceptable.

vdrvOpCodeGetMaxRcvBlockSize
Returns the maximum receive block size that the Serial Manager should request from the virtual device. Can be used to implement flow control.
vdrvOpCodeNotifyBytesReadFromQ Tells the virtual device that some number of bytes have been read from the receive queue by the client application. Can be used to implement flow control.

vdrvOpCodeSetDTRAffirmed Asserts or de-asserts the DTR signal.

vdrvOpCodeGetDTRAffirmed Gets the status of the DTR signal.

vdrvOpCodeWaitForConfiguration Waits for USB enumeration to complete. Called from the send and receive functions of the Serial Manager. The driver should have a timeout for how long it waits for enumeration to complete. The driver should return with no error if enumeration has already occurred or has occurred within the driver’s timeout. If the enumeration has not occurred within the driver’s timeout, the driver should return serErrTimeOut.

vdrvOpCodeGetUSBDeviceDescriptor RETRIEVES THE DEVICE DESCRIPTOR OF A USB DRIVER. USED TO GATHER INFORMATION ABOUT THE DEVICE’S CAPABILITIES. IMPLEMENTATION OF THIS OP CODE IS OPTIONAL. IF THE DRIVER CHOOSES TO IMPLEMENT THIS OP CODE, THEN THE DRIVER SHOULD RETURN A POINTER TO THE DEVICE DESCRIPTOR. A DRIVER THAT CHOOSES NOT TO IMPLEMENT THIS OP CODE SHOULD RETURN serErrNotSupported.

vdrvOpCodeGetUSBConfigDescriptor RETRIEVES THE CONFIGURATION DESCRIPTOR OF A USB DRIVER. USED TO GATHER INFORMATION ABOUT THE DEVICE’S CAPABILITIES. IMPLEMENTATION OF THIS OP CODE IS OPTIONAL. IF THE DRIVER CHOOSES TO IMPLEMENT THIS OP CODE, THEN THE DRIVER SHOULD RETURN A POINTER TO THE DEVICE DESCRIPTOR. A DRIVER THAT CHOOSES NOT TO IMPLEMENT THIS OP CODE SHOULD RETURN serErrNotSupported.

vdrvOpCodeEnableIRDA Enable the IrDA mode and power up the IR line drivers.
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vdrvOpCodeDisableIRDA</td>
<td>Disable the IrDA mode and disable the IR line drivers.</td>
</tr>
<tr>
<td>vdrvOpCodeEnableUART</td>
<td>Powers up the UART and the line drivers.</td>
</tr>
<tr>
<td>vdrvOpCodeDisableUART</td>
<td>Powers down the UART and the line drivers.</td>
</tr>
<tr>
<td>vdrvOpCodeRxEnable</td>
<td>Enables the receive FIFO, enables UART interrupts, and does whatever else is necessary to allow the UART to receive data.</td>
</tr>
<tr>
<td>vdrvOpCodeRxDisable</td>
<td>Disables the receive FIFO and UART interrupts and does whatever is needed to prevent the UART from receiving data.</td>
</tr>
<tr>
<td>vdrvOpCodeLineEnable</td>
<td>Enables the main serial line driver for the UART.</td>
</tr>
<tr>
<td>vdrvOpCodeEnableUARTInterrupts</td>
<td>Enables the appropriate UART receive interrupts.</td>
</tr>
<tr>
<td>vdrvOpCodeDisableUARTInterrupts</td>
<td>Disables all UART interrupts.</td>
</tr>
<tr>
<td>vdrvOpCodeSetReceiveQueue</td>
<td>This op code is used by the Serial Manager to set the driver’s receive queue. This control code is called when a driver that has previously been opened as a background port is opened as a fully open bidirectional port.</td>
</tr>
<tr>
<td>vdrvOpCodeSaveState</td>
<td>Invoked when this port is yielded. This is a hook for the driver to save any current state.</td>
</tr>
<tr>
<td>vdrvOpCodeRestoreState</td>
<td>Invoked when the foreground port is closed and this port can become the foreground port.</td>
</tr>
<tr>
<td>vdrvOpCodeSetYieldPortCallback</td>
<td>Set the function to be called if the Serial Manager attempts to open another port when this one is open. This op code is for system use only.</td>
</tr>
</tbody>
</table>
Compatibility

The op codes starting at vdrvOpCodeWaitForConfiguration are defined only if New Serial Manager Feature Set Version 2 is present. The op codes for yieldable ports and custom operations are defined only if both 4.0 New Feature Set is present as well.

Driver Constants

Driver Version Constants

The driver version constants specify which version of the driver API is implemented by this driver. The DrvEntryPointProcPtr function passes this value back to the Serial Manager in the drvrVersion field of the DrvrInfoType function.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kDrvrVersion</td>
<td>4</td>
<td>The latest version of the API.</td>
</tr>
<tr>
<td>kDrvrVersion3</td>
<td>3</td>
<td>The version of the driver API that corresponds to New Serial Manager Feature Set Version 1 (which ships with roughly Palm OS® 3.3 up to Palm OS 4.0).</td>
</tr>
<tr>
<td>kDrvrVersion4</td>
<td>4</td>
<td>The version of the driver API that corresponds to New Serial Manager Feature Set Version 2 (which ships with roughly Palm OS 4.0 and higher).</td>
</tr>
</tbody>
</table>
Port Feature Constants

The port feature constants are flags that describe serial hardware capabilities.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>portPhysicalPort</td>
<td>0x00000001</td>
<td>Should be set for a physical port, unset for a virtual port.</td>
</tr>
<tr>
<td>portBkgndModeSupportd</td>
<td>0x00000002</td>
<td>Denotes that this port can be used for background ports. This flag is only applicable to virtual drivers. Background mode support is implied on physical drivers.</td>
</tr>
<tr>
<td>portRS232Capable</td>
<td>0x00000004</td>
<td>Set if this hardware has an RS-232 port.</td>
</tr>
<tr>
<td>portIRDACapable</td>
<td>0x00000008</td>
<td>Set if this hardware has an IR port and supports IrDA mode.</td>
</tr>
<tr>
<td>portCradlePort</td>
<td>0x00000010</td>
<td>Set if this hardware controls the cradle port.</td>
</tr>
<tr>
<td>portExternalPort</td>
<td>0x00000020</td>
<td>Set if this hardware port is external or on a memory card.</td>
</tr>
<tr>
<td>portModemPort</td>
<td>0x00000040</td>
<td>Set if this hardware communicates with a modem.</td>
</tr>
<tr>
<td>portCncMgrVisible</td>
<td>0x00000080</td>
<td>Set if this serial port’s name is to be displayed in the Connection panel.</td>
</tr>
<tr>
<td>portConsolePort</td>
<td>0x00000100</td>
<td>Denotes this hardware controls the console port.</td>
</tr>
<tr>
<td>portUSBCapable</td>
<td>0x00000200</td>
<td>Set if this hardware has a USB port.</td>
</tr>
<tr>
<td>portPrivateUse</td>
<td>0x00001000</td>
<td>Set if this driver is for special software and not general applications.</td>
</tr>
</tbody>
</table>

Compatibility

USB support is only available if New Serial Manager Feature Set Version 2 is present.
Virtual Driver-Defined Functions

The functions in this section must be defined by your virtual driver.

**DrvEntryPointProcPtr**

**Purpose**
Entry point for the virtual driver.

**Declared In**
SerialDrvr.h

**Prototype**
```c
Err (*DrvEntryPointProcPtr)(DrvrEntryOpCodeEnum opCode, void *uartData)
```

**Parameters**
- `opCode`  
  Entry function code.
- `uartData`  
  Pointer to data specific to `opCode`.

**Result**
- `errNone`  
  No error.
- `-1`  
  The op code is invalid or the hardware could not be found.

**Comments**
This function’s purpose is based on the value of the `opCode` parameter. The three possible codes are `drvrEntryGetUartFeatures`, `drvrEntryGetDrvrFuncts`, and `drvrEntryGetUartFtrsNEntries`.

`DrvEntryPoint` is called with the `drvrEntryGetUartFeatures` code when the Serial Manager is installed into the system at boot time and is looking for all installed drivers. When this op code is set, the `uartData` pointer points to a `DrvrInfoType` structure. This function does not allocate the structure, it just fills in the fields with information.

This function should check to make sure the associated serial device can operate under the current OS and system settings. If the hardware cannot be found, the function should leave the `DrvrInfoType` struct untouched and return a -1 error.

The driver needs to supply a string that describes the port it manages. This string is displayed to the user in the Connection panel and is returned by the `SrmGetDeviceInfo` function. To set
this string, copy it into the portDesc field of the DrvrInfoType structure. Alternatively, you can supply this string in a driver resource of type 'ISTR' and id kPortDescStrID.

If the DrvrInfoType structure has a positive value in the multipleEnumerations field upon return, the Serial Manager defines one port for each entry in the driver table. The DrvEntryPoint function is called again, this time with the drvrEntryGetUartFtrsNEntries code. The uartData pointer points to a new DrvrInfoType structure whose multipleEnumerations field indicates which port is to be defined. The function should supply all information specific to this port.

DrvEntryPoint is called with the drvrEntryGetDrvrFuncts code when a virtual port is opened. The uartData pointer points to a VdrvAPIType structure and DrvEntryPoint must fill in the fields of this structure with appropriate function pointers.

**Compatibility**
Implemented only if New Serial Manager Feature Set Version 1 is present.

The drvrEntryGetUartFtrsNEntries is only supported if New Serial Manager Feature Set Version 2 is present. This function is fully backwards compatible. Passing 0 for the multipleEnumerations field defines a single port for the driver.

### VdrvCloseProcPtr

**Purpose**
Handles all activities needed to close the virtual device.

**Declared In**
SerialDrvr.h

**Prototype**
Err (*VdrvCloseProcPtr) (VdrvDataPtr drvrData)

**Parameters**
- -> drvrData Pointer to the driver’s private global area.

**Result**
errNone No error.
Virtual Drivers
Virtual Driver-Defined Functions

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

VdrvControlProcPtr

Purpose Extends the SrmControl function to the level of the virtual device.

Declared In SerialDrvr.h

Prototype Err (*VdrvControlProcPtr) (VdrvDataPtr drvrData, VdrvCtlOpCodeEnum controlCode, void *controlData, UInt16 *controlDataLen)

Parameters
- drvrData Pointer to the driver’s private global area.
- controlCode Control function op code. One of the op codes listed in the VdrvCtlOpCodeEnum type.
- controlData Pointer to data for the specified control function.
- controlDataLen Pointer to length of control data being passed in or out.

Result errNone No error.
serErrNotSupported controlCode not supported.
serErrBadParam controlData or controlDataLen is bad.

Comments This function should support the op codes listed in the VdrvCtlOpCodeEnum type. If this function does not support an op code, it must return the serErrNotSupported error code for that op code.

Table 64.1 shows what is passed for the controlData and controlDataLen parameters for each of the control codes that use them. Control codes not listed do not use these parameters.
### Table 64.1 VDrvControlProcPtr Parameters

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vdrvOpCodeSetBaudRate</td>
<td>-&gt; controlData = Pointer to Int32 (baud rate),</td>
</tr>
<tr>
<td></td>
<td>-&gt; controlDataLen = Pointer to sizeof(Int32).</td>
</tr>
<tr>
<td>vdrvOpCodeSetSettingsFlags</td>
<td>-&gt; controlData = Pointer to UInt32 (bitfield; see Serial Settings Constants)</td>
</tr>
<tr>
<td></td>
<td>-&gt; controlDataLen = Pointer to sizeof(UInt32).</td>
</tr>
<tr>
<td>vdrvOpCodeFIFOCount</td>
<td>-&gt; controlData = Pointer to Int16, which contains the number of bytes in the FIFO.</td>
</tr>
<tr>
<td></td>
<td>-&gt; controlDataLen = Pointer to sizeof(Int16).</td>
</tr>
<tr>
<td>vdrvOpCodeGetOptTransmitSize</td>
<td>&lt;- controlData = Pointer to Int32 (buffer size),</td>
</tr>
<tr>
<td></td>
<td>&lt;- controlDataLen = Pointer to sizeof(Int32).</td>
</tr>
<tr>
<td></td>
<td>Return the optimum buffer size for sending data, or 0 to specify any buffer size is acceptable.</td>
</tr>
<tr>
<td>vdrvOpCodeGetMaxRcvBlockSize</td>
<td>&lt;- controlData = Pointer to Int32 (block size),</td>
</tr>
<tr>
<td></td>
<td>&lt;- controlDataLen = Pointer to sizeof(Int32).</td>
</tr>
<tr>
<td></td>
<td>Return the maximum block size that the Serial Manager should request from the virtual device.</td>
</tr>
<tr>
<td>vdrvOpCodeNotifyBytes ReadFromQ</td>
<td>-&gt; controlData = Pointer to Int32 (number of bytes read),</td>
</tr>
<tr>
<td></td>
<td>-&gt; controlDataLen = Pointer to sizeof(Int32).</td>
</tr>
<tr>
<td>vdrvOpCodeSetDTRAsserted</td>
<td>-&gt; controlData = Pointer to Boolean indicating whether to enable or disable DTR.</td>
</tr>
<tr>
<td></td>
<td>-&gt; controlDataLen = Pointer to sizeof(Boolean).</td>
</tr>
</tbody>
</table>
VdrvControlCustomProcPtr

**Purpose**
Extends the SrmCustomControl function to the level of the virtual device.

**Declared In**
SerialDrvr.h

**Prototype**

```c
Err (*VdrvControlCustomProcPtr)(VdrvDataPtr drvrData, UInt16 opCode, UInt32 creator, void *controlData, void *controlDataLenP)
```

**Parameters**
- `drvrData`  
  Pointer to the driver’s private global area.
- `controlCode`  
  Control function op code.
- `creator`  
  Creator ID of the driver that defines the op code. The combination of creator ID and op code uniquely identifies the operation to be performed.
- `controlData`  
  Pointer to data for the specified control function.

**Compatibility**
Implemented only if **New Serial Manager Feature Set Version 1** is present.

---

**Table 64.1 VDrvControlProcPtr Parameters (continued)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vdrvOpCodeGetDTRAsserted</td>
<td>`&lt;-&gt; controlData = Pointer to Boolean indicating whether DTR is enabled.</td>
</tr>
<tr>
<td></td>
<td>`&lt;-&gt; controlDataLen = Pointer to sizeof(Boolean)</td>
</tr>
<tr>
<td>vdrvOpCodeUserDef</td>
<td>`&lt;-&gt; controlData = Pointer from SrmControl (user-defined data),</td>
</tr>
<tr>
<td></td>
<td>`&lt;-&gt; controlDataLen = Pointer to sizeof(Int32).</td>
</tr>
</tbody>
</table>
controlDataLen

Pointer to length of control data being passed in or out.

Result  errNone            No error.
        serErrNotSupported  controlCode not supported.
        serErrBadParam     controlData or controlDataLen is bad.

Comments This function is a mechanism for a virtual driver to create control codes specific to that driver, allowing for the support of new technologies that have interfaces through the Serial Manager.

Compatibility Implemented only if both New Serial Manager Feature Set Version 2 and 4.0 New Feature Set are present.

VdrvOpenProcPtr

Purpose Initializes the virtual device to begin communication.

Declared In SerialDrvr.h

Prototype  Err (*VdrvOpenProcPtr) (VdrvDataPtr *drvrData, UInt32 baudRate, DrvrHWRcvQPtr rcvQP)

Parameters  <> drvrData  Pointer to a pointer to the driver’s private global area (allocated by this function). A pointer to this private global area is passed to the other virtual driver functions.
            -> baudRate  Initial baud rate setting.
            -> rcvQP  Pointer to the driver’s receive queue buffer structure. For details on the fields, see DrvrRcvQType.

Result  errNone            No error.
**Virtual Drivers**

*Virtual Driver-Defined Functions*

---

**Comments**

This function must allocate and initialize any global variables (and pass back a pointer to a pointer to them in `drvrDataP`), do any set-up necessary for communicating with other software, and save the `rcvQP` pointer since it will need the functions and pointers to structures enclosed within to be able to save received data into the Serial Manager’s receive queue.

**Compatibility**

Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

---

**VdrvOpenProcV4Ptr**

**Purpose**

Initializes the virtual device to begin communication.

**Declared In**

`SerialDrvr.h`

**Prototype**

```c
Err (*VdrvOpenProcV4Ptr) (VdrvDataPtr *drvrData,
VdrvConfigPtr configP, DrvrHWRcvQPtr rcvQP)
```

**Parameters**

- `<-> drvrData` Pointer to a pointer to the driver’s private global area (allocated by this function). A pointer to this private global area is passed to the other virtual driver functions.

- `-> configP` Pointer to the configuration structure specifying the port’s properties. See `VdrvConfigType`.

- `-> rcvQP` Pointer to the driver’s receive queue buffer structure. For details on the fields, see `DrvrRcvQType`.

**Result**

`errNone` No error.

**Comments**

This function must allocate and initialize any global variables (and pass back a pointer to a pointer to them in `drvrDataP`), do any set-up necessary for communicating with other software, and save the `rcvQP` pointer since it will need the functions and pointers to structures enclosed within to be able to save received data into the Serial Manager’s receive queue.
**Compatibility**

Implemented only if both [New Serial Manager Feature Set Version 2](#) and [4.0 New Feature Set](#) are present.

---

**VdrvStatusProcPtr**

**Purpose**

Returns virtual device status.

**Declared In**

SerialDrvr.h

**Prototype**

`UInt16 (*VDrvStatusProcPtr) (VdrvDataPtr drvrData)`

**Parameters**

- `drvrData` Pointer to the driver’s private global area.

**Result**

An unsigned long bitfield denoting the status of the virtual device, but only if the virtual device is emulating hardware. The individual bit flags are described in the `DrvStatusEnum` type.

**Comments**

Generally, status is returned only to the client application using the virtual device. The Serial Manager does not use status information from virtual devices.

**Compatibility**

Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

---

**VdrvWriteProcPtr**

**Purpose**

Writes a block of bytes.

**Declared In**

SerialDrvr.h

**Prototype**

`UInt32 (*VdrvWriteProcPtr) (VdrvDataPtr drvrDataP, void *bufP, UInt32 size, Err *errP)`

**Parameters**

- `drvrDataP` Pointer to the driver’s private global area.
- `bufP` Pointer to buffer containing the data to be written to the virtual device.
- `size` Number of bytes in the buffer `bufP`. 
Serial Manager Queue Functions

The functions in this section are supplied by the Serial Manager to the virtual driver through the `DrvrRcvQType` passed to the `VdrvOpenProcPtr` function.

GetSizeProcPtr

**Purpose**

Returns the total size of the Serial Manager’s receive queue.

**Declared In**

SerialDrv.h

**Prototype**

```c
typedef UInt32 (*GetSizeProcPtr) (void *theQ)
```

**Parameters**

- `theQ` Pointer to the receive queue.

**Result**

Size in bytes of the Serial Manager’s receive queue.

**Comments**

This function is useful for implementing flow control.

**Compatibility**

Implemented only if [New Serial Manager Feature Set Version 1](#) is present.
**GetSpaceProcPtr**

**Purpose**  Returns the available space in the Serial Manager’s receive queue.

**Declared In**  SerialDrvr.h

**Prototype**  
```c
typedef UInt32 (*GetSpaceProcPtr) (void *theQ)
```

**Parameters**  
- `theQ`  Pointer to the receive queue.

**Result**  Size in bytes of the available space in the Serial Manager’s receive queue.

**Comments**  This function is useful for implementing flow control.

**Compatibility**  Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

---

**SignalCheckPtr**

**Purpose**  Check the queue to see if the semaphore needs to be signalled.

**Declared In**  SerialDrvr.h

**Prototype**  
```c
typedef void (*SignalCheckPtr) (void *theQ,
                                 UInt16 lineErrsP)
```

**Parameters**  
- `theQ`  Pointer to the receive queue.
- `lineErrsP`  Any serial line errors received should be reported here.

**Result**  Returns nothing.

**Comments**  This function signals that there is data to be received without writing anything to the receive queue. The `WriteByteProcPtr` and `WriteBlockProcPtr` functions also signal that there is data to be received after they have written the data to the queue.
Virtual Drivers
Serial Manager Queue Functions

Compatibility
Implemented only if New Serial Manager Feature Set Version 2 is present.

WriteBlockProcPtr

Purpose
Writes a block of bytes to the Serial Manager’s receive queue.

Declared In
SerialDrvr.h

Prototype
typedef Err (*WriteBlockProcPtr) (void *theQ, UInt8 *bufP, UInt16 size, UInt16 lineErrs)

Parameters
- theQ Pointer to the receive queue.
- bufP Pointer to the buffer holding bytes to be written.
- size Size of bufP.
- lineErrs Any serial line errors received should be reported here.

Result
errNone No error.
serErrLineErr There was a software overrun line error.

Compatibility
Implemented only if New Serial Manager Feature Set Version 1 is present.

WriteByteProcPtr

Purpose
Writes one byte to the Serial Manager’s receive queue.

Declared In
SerialDrvr.h

Prototype
typedef Err (*WriteByteProcPtr) (void *theQ, UInt8 theByte, UInt16 lineErrs)

Parameters
- theQ Pointer to the receive queue.
- theByte The byte to be written to the queue.
**Virtual Drivers**  
*Serial Manager Queue Functions*

<table>
<thead>
<tr>
<th>-&gt; lineErrs</th>
<th>Any serial line errors received should be reported here.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Result</strong></td>
<td><strong>errNone</strong></td>
</tr>
<tr>
<td></td>
<td><strong>serErrLineErr</strong></td>
</tr>
<tr>
<td><strong>Compatibility</strong></td>
<td>Implemented only if <a href="#">New Serial Manager Feature Set Version 1</a> is present.</td>
</tr>
</tbody>
</table>
Serial Manager

This chapter provides reference material for the Serial Manager API:

- Serial Manager Data Structures
- Serial Manager Constants
- Serial Manager Functions
- Serial Manager Application-Defined Functions

The header file SerialMgr.h declares the Serial Manager API. The file SystemResources.h defines some serial port constants. For more information on the Serial Manager, see the chapter “Serial Communication” on page 89 of the Palm OS Programmer’s Companion, vol. II, Communications.

Serial Manager Data Structures

DeviceInfoType

The DeviceInfoType structure defines information about a serial device. This structure is returned by the SrmGetDeviceInfo function.

```c
typedef struct DeviceInfoType {
    UInt32 serDevCreator;
    UInt32 serDevFtrInfo;
    UInt32 serDevMaxBaudRate;
    UInt32 serDevHandshakeBaud;
    Char *serDevPortInfoStr;
    UInt8 reserved[8];
} DeviceInfoType;

typedef DeviceInfoType *DeviceInfoPtr;
```
Value Descriptions

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serDevCreator</td>
<td>Four-character creator ID for serial driver.</td>
</tr>
<tr>
<td>serDevPtrInfo</td>
<td>Flags defining features of this serial hardware. See Serial Capabilities</td>
</tr>
<tr>
<td>serDevMaxBaudRate</td>
<td>Constants for a description of these flags.</td>
</tr>
<tr>
<td>serDevHandshakeBaud</td>
<td>Hardware handshaking is recommended for baud rates over this rate.</td>
</tr>
<tr>
<td>serDevPortInfoStr</td>
<td>Description of serial hardware device or virtual device.</td>
</tr>
</tbody>
</table>

SrmCtlEnum

The SrmCtlEnum enumerated type specifies a serial control operation. Specify one of these enumerated types for the op parameter to the SrmControl call.

```c
typedef enum SrmCtlEnum {
    srmCtlFirstReserved = 0,
    srmCtlSetBaudRate,
    srmCtlGetBaudRate,
    srmCtlSetFlags,
    srmCtlGetFlags,
    srmCtlSetCtsTimeout,
    srmCtlGetCtsTimeout,
    srmCtlStartBreak,
    srmCtlStopBreak,
    srmCtlStartLocalLoopback,
    srmCtlStopLocalLoopback,
    srmCtlIrDAEnable,
    srmCtlIrDADisable,
    srmCtlRxEnable,
    srmCtlRxDisable,
    srmCtlEmuSetBlockingHook,
    srmCtlUserDef,
} SrmCtlEnum;
```
srmCtlGetOptimalTransmitSize,
srmCtlSetDTRAsserted,
srmCtlGetDTRAsserted,
srmCtlSetYieldPortCallback,
srmCtlSetYieldPortRefCon,
srmCtlSystemReserved = 0x7000
srmCtlCustom = 0x8000,
srmCtlLAST
} SrmCtlEnum;

Value Descriptions

srmCtlSetBaudRate    Sets the current baud rate for the serial hardware.
srmCtlGetBaudRate    Gets the current baud rate for the serial hardware.
srmCtlSetFlags      Sets the current flag settings for the serial hardware. Specify flags from the set described in Serial Settings Constants.
srmCtlGetFlags      Gets the current flag settings for the serial hardware.
srmCtlSetCtsTimeout Sets the current CTS timeout value for hardware handshaking.
srmCtlGetCtsTimeout Gets the current CTS timeout value for hardware handshaking.
srmCtlStartBreak    Turn RS-232 break signal on. Caller is responsible for turning this signal on and off and insuring it is on long enough to generate a viable break.
srmCtlStopBreak     Turn RS-232 break signal off.
srmCtlStartLocalLoopback Start local loopback test.
srmCtlStopLocalLoopback Stop local loopback test.
srmCtlIrDAEnable

Enable IrDA connection on this serial port.

NOTE: You cannot enable an IrDA connection on a VZ processor.

srmCtlIrDADisable

Disable IrDA connection on this serial port.

srmCtlRxEnable

Enable receiver (for IrDA).

srmCtlRxDisable

Disable receiver (for IrDA).

srmCtlEmuSetBlockingHook

Set a blocking hook routine for emulation mode only. Not supported on the actual device.

srmCtlUserDef

This is a user-defined function that third-party hardware developers can use to set or retrieve hardware-specific information from the serial driver. This op code invokes the driver’s corresponding control function with its user-defined op code and the parameters are passed directly through to the serial driver. A serial driver that does not handle this function returns a ser ErrBadParam error.

The srmCtlUserDef op code is superseded by defining a custom op code if New Serial Manager Feature Set Version 2 is present.

srmCtlGetOptimalTransmitSize

Ask the port for the most efficient buffer size for transmitting data packets. This op code returns an error (buffering not necessary), 0 (buffering requested, but application can choose buffer size), or a number greater than 0 (recommended buffer size).

srmCtlSetDTRAsserted

Enabled or disable the DTR signal. This is not supported by all hardware.

srmCtlGetDTRAsserted

Ask the port whether the DTR signal is enabled or disabled.
srmCtlSetYieldPortCallback Set the function to be called if the Serial Manager attempts to open another port when this one is open. This op code is for system use only.

srmCtlSetYieldPortRefCon Data to pass to the yield port callback function. System use only.

srmCtlSystemReserved Reserves op codes between 0x7000 and 0x8000 for system use.

srmCtlCustom Reserves op codes greater than 0x8000 for driver-specific use.

**Compatibility** Custom control op codes are only supported if both [New Serial Manager Feature Set Version 2](#) and [4.0 New Feature Set](#) are present.

### SrmOpenConfigType

The `SrmOpenConfigType` structure specifies parameters for opening a serial port. This structure is passed as a parameter to `SrmExtOpen`.

```c
typedef struct SrmOpenConfigType {
    UInt32 baud;
    UInt32 function;
    MemPtr drvrDataP;
    UInt16 drvrDataSize;
    UInt32 sysReserved1;
    UInt32 sysReserved2;
} SrmOpenConfigType;
```
Field Descriptions

**baud**
Baud rate at which to open the connection. Serial drivers that do not require baud rates ignore this field.

**function**
The reason why the port was opened. Specify the creator ID of the application that is opening the port or one of the following values:

- **serFncUndefined**
  Undefined function. This is the default value for this field.

- **serFncPPPSession**
  The connection is to be used for the PPP interface.

- **serFncSLIPSession**
  The connection is to be used for the SLIP session.

- **serFncDebugger**
  The connection is to be used for a debugging session.

- **serFncHotSync**
  The connection is to be used for a HotSync operation.

- **serFncConsole**
  The connection is to the debugging console.

- **serFncTelephony**
  The connection is to the telephony library.

The **function** field is used by protocols such as USB and Bluetooth that perform different setup tasks based on which type of application is using them. RS-232 drivers ignore this parameter.

**drvrDataP**
Pointer to a driver-specific data block.

**drvrDataSize**
The size of the data block pointed to by **drvrDataP**.
Serial Manager Constants

Port Constants
When you specify the port to open in the `SrmOpen`, `SrmOpenBackground`, `SrmExtOpen`, or `SrmExtOpenBackground` call, you can use either a logical port constant, physical port constant, or a virtual port constant, but it is highly recommended that you use a logical port constant wherever possible.

### Logical Serial Port Constants
These constants specify the logical port names.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>serPortLocalHotSync</code></td>
<td>0x8000</td>
<td>The physical HotSync port. The Serial Manager automatically detects whether this port is USB or RS-232.</td>
</tr>
<tr>
<td><code>serPortCradlePort</code></td>
<td>0x8000</td>
<td>Cradle port. The Serial Manager automatically detects whether this port is USB or RS-232. Most applications should specify this as the port.</td>
</tr>
<tr>
<td><code>serPortIrPort</code></td>
<td>0x8001</td>
<td>The IR port. This is a raw IrDA port with no protocol support.</td>
</tr>
<tr>
<td><code>serPortConsolePort</code></td>
<td>0x8002</td>
<td>The debug console port, either USB or RS-232. USB is preferred where both are available.</td>
</tr>
</tbody>
</table>
Serial Manager
Serial Manager Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serPortCradleRS232Port</td>
<td>0x8003</td>
<td>Port for the RS-232 cradle. Specify this port if you want to ensure that your application uses RS-232 communications only.</td>
</tr>
<tr>
<td>serPortCradleUSBPort</td>
<td>0x8004</td>
<td>Port for the USB cradle. Specify this port if you want to ensure that your application uses USB communications only.</td>
</tr>
</tbody>
</table>

Compatibility
USB ports are only supported if New Serial Manager Feature Set Version 2 is present.

Physical Serial Port Constants
The physical port constants specify 4-character constants that reference the physical hardware of the device. Doing so is not recommended because the hardware they reference may not exist on a particular device.

<table>
<thead>
<tr>
<th>Physical port</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysFileCUart328</td>
<td>'u328'</td>
<td>Cradle port using the 68328 UART. This port can be switched between RS232 and IrDA mode using the SrmControl call.</td>
</tr>
<tr>
<td>sysFileCUart328EZ</td>
<td>'u8EZ'</td>
<td>Cradle port using the 68328EZ UART. This port can also be switched between RS232 and IrDA mode.</td>
</tr>
<tr>
<td>sysFileCUart650</td>
<td>'u650'</td>
<td>Specifies the IR port on the upgrade card for Palm Personal or Palm Professional devices. This gives you a raw IR port like calling SrmControl does, but it only exists on devices that have the upgrade card.</td>
</tr>
</tbody>
</table>

Virtual Serial Port Constants
The virtual port constants specify 4-character constants that identify virtual ports, simulating a hardware interface. Virtual ports are not tied to specific hardware.
### Serial Manager

#### Serial Manager Constants

<table>
<thead>
<tr>
<th>Physical port</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysFileCVirtIrComm</td>
<td>'ircm'</td>
<td>A virtual serial cable over an IrDA link using the IRComm protocol. It can only be used to talk to another IRComm device.</td>
</tr>
<tr>
<td>sysFileCVirtRfComm</td>
<td>'rfcm'</td>
<td>RFCOMM (Bluetooth) virtual port plug-in.</td>
</tr>
<tr>
<td>sysFileCBtConnectPanelHelper</td>
<td>'btcp'</td>
<td>Bluetooth Connection Panel helper application.</td>
</tr>
</tbody>
</table>

**Compatibility**

All virtual port constants other than `sysFileCVirtIrComm` are only defined if both [New Serial Manager Feature Set Version 2](#) and [4.0 New Feature Set](#) are present.

### Serial Capabilities Constants

The serial capabilities constant flags describe serial hardware capabilities. These flags are set in the `serDevFtrInfo` field of the `DeviceInfoType` structure.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serDevCradlePort</td>
<td>0x00000001</td>
<td>Serial hardware controls RS-232 serial from cradle connector of Palm device.</td>
</tr>
<tr>
<td>serDevRS232Serial</td>
<td>0x00000002</td>
<td>Serial hardware has RS-232 line drivers.</td>
</tr>
<tr>
<td>serDevIRDACapable</td>
<td>0x00000004</td>
<td>Serial hardware has IR line drivers and generates IrDA mode serial signals.</td>
</tr>
<tr>
<td>serDevModemPort</td>
<td>0x00000008</td>
<td>Serial hardware drives modem connection.</td>
</tr>
<tr>
<td>serDevCncMgrVisible</td>
<td>0x00000010</td>
<td>Serial device port name string is to be displayed in the Connection panel.</td>
</tr>
</tbody>
</table>
Serial Manager
Serial Manager Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serDevConsolePort</td>
<td>0x00000020</td>
<td>Serial device is the default console port.</td>
</tr>
<tr>
<td>serDevUSBCapable</td>
<td>0x00000040</td>
<td>Serial hardware controls USB serial from cradle connector of Palm device.</td>
</tr>
</tbody>
</table>

Compatibility  USB ports are only supported if New Serial Manager Feature Set Version 2 is present.

Serial Settings Constants

The serial settings constants identify bit flags that correspond to various serial hardware settings. Use SrmControl with the op code srmCtlSetFlags to control which settings are used.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srmSettingsFlagStopBitsM</td>
<td>0x00000001</td>
<td>Mask for stop bits field</td>
</tr>
<tr>
<td>srmSettingsFlagStopBits1</td>
<td>0x00000000</td>
<td>1 stop bit (default)</td>
</tr>
<tr>
<td>srmSettingsFlagStopBits2</td>
<td>0x00000001</td>
<td>2 stop bits</td>
</tr>
<tr>
<td>srmSettingsFlagParityOnM</td>
<td>0x00000002</td>
<td>Mask for parity on</td>
</tr>
<tr>
<td>srmSettingsFlagParityEvenM</td>
<td>0x00000004</td>
<td>Mask for parity even</td>
</tr>
<tr>
<td>srmSettingsFlagXonXoffM</td>
<td>0x00000008</td>
<td>Mask for Xon/Xoff flow control (not implemented)</td>
</tr>
<tr>
<td>srmSettingsFlagRTSAutoM</td>
<td>0x00000010</td>
<td>Mask for RTS receive flow control. This is the default.</td>
</tr>
<tr>
<td>srmSettingsFlagCTSAutoM</td>
<td>0x00000020</td>
<td>Mask for CTS transmit flow control</td>
</tr>
<tr>
<td>srmSettingsFlagBitsPerCharM</td>
<td>0x000000C0</td>
<td>Mask for bits per character</td>
</tr>
<tr>
<td>srmSettingsFlagBitsPerChar5</td>
<td>0x00000000</td>
<td>5 bits per character</td>
</tr>
<tr>
<td>srmSettingsFlagBitsPerChar6</td>
<td>0x00000040</td>
<td>6 bits per character</td>
</tr>
</tbody>
</table>
### Serial Manager Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srmSettingsFlagBitsPerChar7</td>
<td>0x00000080</td>
<td>7 bits per character</td>
</tr>
<tr>
<td>srmSettingsFlagBitsPerChar8</td>
<td>0x000000C0</td>
<td>8 bits per character (default)</td>
</tr>
<tr>
<td>srmSettingsFlagFlowControlIn</td>
<td>0x00000100</td>
<td>Protect the receive buffer from software overruns. When this flag and srmSettingsFlagRTSAMtom are set, which is the default case, it causes the Serial Manager to assert RTS to prevent the transmitting device from continuing to send data when the receive buffer is full. Once the application receives data from the buffer, RTS is de-asserted to allow data reception to resume. Note that this feature effectively prevents software overrun line errors but may also cause CTS timeouts on the transmitting device if the RTS line is asserted longer than the defined CTS timeout value.</td>
</tr>
<tr>
<td>srmSettingsFlagRTSInactive</td>
<td>0x00000200</td>
<td>If this flag is set and srmSettingsFlagRTSAMtom is not set, RTS is held in the inactive (flow off) state forever.</td>
</tr>
</tbody>
</table>
Serial Manager
Serial Manager Constants

Status Constants
The status constants identify bit flags that correspond to the status of serial signals. They can be returned by the SrmGetStatus function.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srmStatusCtsOn</td>
<td>0x00000001</td>
<td>CTS line is active.</td>
</tr>
<tr>
<td>srmStatusRtsOn</td>
<td>0x00000002</td>
<td>RTS line is active.</td>
</tr>
<tr>
<td>srmStatusDsrOn</td>
<td>0x00000004</td>
<td>DSR line is active.</td>
</tr>
<tr>
<td>srmStatusBreakSigOn</td>
<td>0x00000008</td>
<td>Break signal is active.</td>
</tr>
</tbody>
</table>

Line Error Constants
The line error constants identify bit flags that correspond to the line errors that may occur on the port. They can be returned by the SrmGetStatus function.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serLineErrorParity</td>
<td>0x0001</td>
<td>Parity error</td>
</tr>
<tr>
<td>serLineErrorHWOverrun</td>
<td>0x0002</td>
<td>Hardware overrun</td>
</tr>
<tr>
<td>serLineErrorFraming</td>
<td>0x0004</td>
<td>Framing error</td>
</tr>
<tr>
<td>serLineErrorBreak</td>
<td>0x0008</td>
<td>Break signal asserted</td>
</tr>
<tr>
<td>serLineErrorHShake</td>
<td>0x0010</td>
<td>Line handshake error</td>
</tr>
<tr>
<td>serLineErrorSWOverrun</td>
<td>0x0020</td>
<td>Software overrun</td>
</tr>
<tr>
<td>serLineErrorCarrierLost</td>
<td>0x0040</td>
<td>Carrier detect signal dropped</td>
</tr>
</tbody>
</table>
Serial Manager Functions

**SrmClearErr**

**Purpose**
Clears the port of any line errors.

**Declared In**
SerialMgr.h

**Prototype**
Err SrmClearErr (UInt16 portId)

**Parameters**
- `portID`
  Port ID returned from `SrmOpen` or `SrmExtOpen`.

**Result**
This function returns the following error codes:
- `errNone`
  No error.
- `serErrNotSupported`
  The port is not the foreground port.

**Compatibility**
Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

**SrmClose**

**Purpose**
Closes a serial port and makes it available to other applications, regardless of whether the port is a foreground or background port.

**Declared In**
SerialMgr.h

**Prototype**
Err SrmClose (UInt16 portId)

**Parameters**
- `portId`
  Port ID for port to be closed.

**Result**
This function returns the following error codes:
- `errNone`
  No error.
- `serErrBadPort`
  This port doesn’t exist.
Serial Manager
Serial Manager Functions

serErrNotOpen The serial port is not open.
serErrNoDevicesAvail No serial devices could be found.

Comments
If a foreground port is being closed and a background port exists, the background will have access to the port as long as another foreground port is not opened.

If a foreground port is being closed and a yielded port exists, the yielded port will have access to the port as long as it does not yield to the opening of another foreground port. If there are both a yielded port and a background port for the foreground port being closed, the yielded port takes precedence over the background port.

Compatibility
Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmOpen, SrmOpenBackground

SrmControl

Purpose Performs a serial control function.

Declared In SerialMgr.h

Prototype Err SrmControl (UInt16 portId, UInt16 op, void *valueP, UInt16 *valueLenP)

Parameters
-> portID Port ID returned from SrmOpen or SrmExtOpen.
-> op Control operation to perform. Specify one of the SrmCtlEnum enumerated types.
<> valueP Pointer to a value to use for the operation. See Comments for details.
<> valueLenP Pointer to the size of *valueP. See Comments for details.

Result This function returns the following error codes:
Serial Manager
Serial Manager Functions

errNone No error.
serErrBadParam An invalid op code was specified.
serErrBadPort This port doesn’t exist.
serErrNotOpen The serial port is not open.
serErrNoDevicesAvail No serial devices could be found.
serErrNotSupported The specified op code is not supported in the current configuration.

Comments Table 65.1 shows what to pass for the valueP and valueLenP parameters for each of the operation codes. Control codes not listed do not use these parameters. See SrmCtlEnum for a complete list of control codes.

Table 65.1 SrmControl Parameters

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>srmCtlSetBaudRate</td>
<td>-&gt; valueP = Pointer to Int32 (baud rate)</td>
</tr>
<tr>
<td></td>
<td>-&gt; valueLenP = Pointer to sizeof (Int32)</td>
</tr>
<tr>
<td>srmCtlGetBaudRate</td>
<td>&lt;- valueP = Pointer to Int32 (baud rate)</td>
</tr>
<tr>
<td></td>
<td>&lt;- valueLenP = Pointer to Int16</td>
</tr>
<tr>
<td>srmCtlSetFlags</td>
<td>-&gt; valueP = Pointer to Uint32 (bitfield; see Serial Settings Constants)</td>
</tr>
<tr>
<td></td>
<td>-&gt; valueLenP = Pointer to sizeof (Uint32)</td>
</tr>
<tr>
<td>srmCtlGetFlags</td>
<td>&lt;- valueP = Pointer to Uint32 (bitfield)</td>
</tr>
<tr>
<td></td>
<td>&lt;- valueLenP = Pointer to Int16</td>
</tr>
<tr>
<td>srmCtlSetCtsTimeout</td>
<td>-&gt; valueP = Pointer to Int32 (timeout value)</td>
</tr>
<tr>
<td></td>
<td>-&gt; valueLenP = Pointer to sizeof (Int32)</td>
</tr>
<tr>
<td>srmCtlGetCtsTimeout</td>
<td>&lt;- valueP = Pointer to Int32 (timeout value)</td>
</tr>
<tr>
<td></td>
<td>&lt;- valueLenP = Pointer to Int16</td>
</tr>
</tbody>
</table>
### Serial Manager

*Serial Manager Functions*

---

#### Table 65.1 SrmControl Parameters *(continued)*

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>srmCtlUserDef</td>
<td><code>&lt;&gt; valueP = Pointer passed to the serial or virtual driver</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;&gt; valueLenP = Pointer to sizeof(Int32)</code></td>
</tr>
<tr>
<td></td>
<td>For a serial driver, these pointers are passed to the driver’s control function and they contain that functions return values (if any) upon return.</td>
</tr>
<tr>
<td>srmCtlGetOptimalTransmitSize</td>
<td><code>&lt;- valueP = Pointer to Int32</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;-valueLenP = Pointer to sizeof(Int32)</code></td>
</tr>
<tr>
<td></td>
<td>If an error is returned by SrmControl, no buffering should be done. If valueP points to zero, buffering is requested, but the transmitting application cannot determine the buffer size. If valueP points to a number &gt; 0, then try to send data in blocks of this number of bytes, as this is the most efficient block size for this particular device.</td>
</tr>
<tr>
<td>srmCtlSetDTRAsserted</td>
<td><code>-&gt; valueP = Pointer to Boolean indicating whether to enable or disable DTR.</code></td>
</tr>
<tr>
<td></td>
<td><code>-&gt; valueLenP = Pointer to sizeof(Boolean)</code></td>
</tr>
<tr>
<td>srmCtlGetDTRAsserted</td>
<td><code>&lt;- valueP = Pointer to Boolean indicating whether DTR is enabled.</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;- valueLenP = Pointer to Int16</code></td>
</tr>
</tbody>
</table>

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

**See Also** [SrmCustomControl](#)
SrmCustomControl

**Purpose**
Performs a custom serial control function.

**Declared In**
SerialMgr.h

**Prototype**
```
Err SrmCustomControl (UInt16 portId,
                     UInt16 opCode, UInt32 creator, void *valueP,
                     UInt16 *valueLenP)
```

**Parameters**
- **portID**
  Port ID returned from `SrmOpen` or `SrmExtOpen`.
- **opCode**
  Control operation to perform. The op code must be greater than `srmCtlCustom`.
- **creator**
  Creator ID of the driver that defines the op code. The combination of creator ID and op code uniquely identifies the operation to be performed.
- **valueP**
  Pointer to a value to use for the operation.
- **valueLenP**
  Pointer to the size of `valueP`.

**Result**
This function returns the following error codes:
- **errNone**
  No error.
- **serErrNotSupported**
  The port is not the foreground port.
- **serErrBadPort**
  This port doesn’t exist.
- **serErrNoOpen**
  The serial port is not open.
- **serErrNoDevicesAvail**
  No serial devices could be found.

**Comments**
This function is a mechanism for a virtual driver to create control codes specific to that driver, allowing for the support of new technologies that have interfaces through the Serial Manager.

This function simply forwards the `opCode` and any `valueP` parameter to the virtual driver for the port. The virtual driver may
return its own error code if the `opCode` or the input in `valueP` is invalid.

**Compatibility** Implemented only if both New Serial Manager Feature Set Version 2 and 4.0 New Feature Set are present.

**See Also** [SrmControl](#)

**SrmExtOpen**

**Purpose** Opens a foreground port connection with the specified configuration.

**Declared In** SerialMgr.h

**Prototype**

```c
Err SrmExtOpen (UInt32 port,
SrmOpenConfigType *configP, UInt16 configSize,
UInt16 *newPortIdP)
```

**Parameters**

- `port` The four-character port name (such as 'ircm' or 'u328') or logical port number to be opened. (See Port Constants.)
- `configP` Pointer to the configuration structure specifying the serial port's properties. See SrmOpenConfigType.
- `configSize` The size of the configuration structure pointed to by `configP`.
- `newPortIdP` Contains the port ID to be passed to other Serial Manager functions.

**Result** This function returns the following error codes:

- `errNone` No error.
- `serErrBadPort` The port parameter does not specify a valid port.
- `serErrBadParam` The `configP` parameter is NULL.
serErrAlreadyOpen
   The Serial Manager already has a port open.

memErrNotEnoughSpace
   There was not enough memory available to open the port.

Comments
   Do not keep the port open any longer than necessary. An open serial port consumes more energy from the device’s batteries.

   The values specified in the configP parameter depend on the type of connection being made. For RS-232 connections, you specify the baud rate but not a purpose. For USB connections, you specify a purpose but not a baud rate.

   A newly opened port has its line errors cleared, the default CTS timeout set (specified by the constant srmDefaultCTSTimeout), a 512-byte receive queue allocated, 1 stop bit, 8 bits per character, RTS enabled, and flow control enabled. To increase the receive queue size, use SrmSetReceiveBuffer. To change the other serial port settings, use SrmControl.

Compatibility
   Implemented only if both New Serial Manager Feature Set Version 2 and 4.0 New Feature Set are present. The SrmExtOpen function replaces the SrmOpen function.

See Also
   SrmOpen, SrmExtOpenBackground
Serial Manager
Serial Manager Functions

SrmExtOpenBackground

Purpose
Opens a port with the specified configuration in the background. Background ports relinquish control when another task opens the port with the SrmOpen or SrmExtOpen call.

Declared In
SerialMgr.h

Prototype
Err SrmExtOpenBackground (UInt32 port, SrmOpenConfigType *configP, Uint16 configSize, Uint16 *newPortIdP)

Parameters
- port
  Physical or logical port number to be opened. See Port Constants for more information.
- configP
  Pointer to the configuration structure specifying the serial port’s properties. See SrmOpenConfigType.
- configSize
  The size of the configuration structure pointed to by configP.
- newPortIdP
  Contains the port ID to be passed to other Serial Manager functions.

Result
This function returns the following error codes:
- errNone: No error.
- serErrAlreadyOpen: This port already has an installed background owner.
- serErrBadPort: This port doesn’t exist.
- serErrNotSupported: This type of port cannot be opened in the background.
- serErrBadParam: The configP parameter is NULL.
- memErrNotEnoughSpace: There was not enough memory available to open the port.
Comments

This function is provided to support tasks that want to use a serial device to receive data only when no other task is using the port.

If a background port is forced to surrender control of the hardware as a result of another task opening a foreground connection, all buffers for the background port are flushed. After this active task closes the port, active control of the port is returned to the background task. Only one task can have background ownership of the port.

Note that background ports have limited functionality: they can only receive data and notify owning clients of what data has been received.

The values specified in the configP parameter depend on the type of connection being made. For RS-232 connections, you specify the baud rate but not a purpose. For USB connections, you specify a purpose but not a baud rate.

Compatibility

Implemented only if both New Serial Manager Feature Set Version 2 and 4.0 New Feature Set are present. The SrmExtOpenBackground function replaces the SrmOpenBackground function.

See Also

SrmOpen, SrmExtOpen

SrmGetDeviceCount

Purpose

Returns the number of available serial devices.

Declared In

SerialMgr.h

Prototype

Err SrmGetDeviceCount (UInt16 *numOfDevicesP)

Parameters

<- numOfDevicesP

    Pointer to address where the number of serial devices is returned.

Result

errNone

No error.
Serial Manager
Serial Manager Functions

**Compatibility**
Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

**See Also**
[SrmGetDeviceInfo](#)

---

**SrmGetDeviceInfo**

**Purpose**
Returns information about a serial device.

**Declared In**
SerialMgr.h

**Prototype**
```c
Err SrmGetDeviceInfo (UInt32 deviceID,
                      DeviceInfoType *deviceInfoP)
```

**Parameters**
- **deviceID**
  ID of serial device to get information for. You can pass a zero-based index (0, 1, 2, ...), a valid port ID returned from [SrmOpen](#) or [SrmExtOpen](#), or a 4-character port name (such as 'u328', 'u650', or 'ircm'). See [Port Constants](#).

- **deviceInfoP**
  Pointer to a [DeviceInfoType](#) structure where information about the device is returned.

**Result**
This function returns the following error codes:
- **errNone**
  No error.
- **serErrBadPort**
  This port doesn’t exist.
- **serErrNoDevicesAvail**
  The Serial Manager cannot find any serial devices.

**Compatibility**
Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

**See Also**
[SrmGetDeviceCount](#)
**SrmGetStatus**

**Purpose**
Returns status information about the serial hardware.

**Declared In**
SerialMgr.h

**Prototype**
Err SrmGetStatus (UInt16 portId,
    UInt32 *statusFieldP, UInt16 *lineErrsP)

**Parameters**
- **portID**
  Port ID returned from SrmOpen or SrmExtOpen.
- **statusFieldP**
  Pointer to address where hardware status information for the port is returned. This is a 32-bit field using the flags described in Status Constants.
- **lineErrsP**
  Pointer to address where the number of line errors for the port is returned. The line error flags are described in Line Error Constants.

**Result**
This function returns the following error codes:
- **errNone**
  No error.
- **serErrBadPort**
  This port doesn’t exist.
- **serErrNotSupported**
  The port is a yielded port.
- **serErrNoDevicesAvail**
  No serial devices could be found.

**Comments**
Typically, SrmGetStatus is called to retrieve the line errors for the port if some of the send and receive functions return a serErrLineErr error code.

**Compatibility**
Implemented only if New Serial Manager Feature Set Version 1 is present.
**SrmOpen**

**Purpose**
Opens a foreground port connection with the specified port name or logical port number.

**Declared In**
SerialMgr.h

**Prototype**
Err SrmOpen (UInt32 port, UInt32 baud, UInt16 *newPortIdP)

**Parameters**
- **port**
The four-character port name or logical port number to be opened. See Port Constants for more information.
- **baud**
Initial baud rate of port.
- **newPortIdP**
Contains the port ID to be passed to other Serial Manager functions.

**Result**
This function returns the following error codes:
- errNone: No error.
- serErrAlreadyOpen: This port already has an installed foreground owner.
- serErrBadPort: This port doesn’t exist.
- memErrNotEnoughSpace: There was not enough memory available to open the port.

**Comments**
Only one application or task may have access to a particular serial port at any time.

Do not keep the port open any longer than necessary. An open serial port consumes more energy from the device's batteries.

**Compatibility**
Implemented only if New Serial Manager Feature Set Version 1 is present.
If **New Serial Manager Feature Set Version 2** is present, the SrmOpen function is replaced by **SrmExtOpen**. SrmOpen is supported for backward compatibility.

**See Also**  
SrmOpenBackground

### SrmOpenBackground

**Purpose**  
Allows a task to open, initialize, and use the port, but always relinquishes control of the port when another task opens the port with the **SrmOpen** call.

**Declared In**  
SerialMgr.h

**Prototype**  
Err SrmOpenBackground (UInt32 port, UInt32 baud, UInt16 *newPortIdP)

**Parameters**

- **port**  
The four-character port name or logical port number to be opened. See [Port Constants](#) for more information.

- **baud**  
Initial baud rate of port.

- **newPortIdP**  
Contains the port ID to be passed to other Serial Manager functions.

**Result**  
This function returns the following error codes:

- **errNone**  
No error.

- **serErrAlreadyOpen**  
This port already has an installed background owner.

- **serErrBadPort**  
This port doesn’t exist.

- **memErrNotEnoughSpace**  
There was not enough memory available to open the port.

**Comments**  
This function is provided to support tasks that want to use a serial device to receive data only when no other task is using the port.
If a background port is forced to surrender control of the hardware as a result of another task opening a foreground connection, all buffers for the background port are flushed. After this active task closes the port, active control of the port is returned to the background task. Only one task can have background ownership of the port.

Note that background ports have limited functionality: they can only receive data and notify owning clients of what data has been received.

**Compatibility**

Implemented only if New Serial Manager Feature Set Version 1 is present.

If New Serial Manager Feature Set Version 2 is present, the SrmOpenBackground function is replaced by SrmExtOpenBackground. SrmOpenBackground is supported for backward compatibility.

**See Also**  
SrmOpen

---

**SrmPrimeWakeupHandler**

**Purpose**
Sets the number of received bytes that triggers a call to the wakeup handler function.

**Declared In**
SerialMgr.h

**Prototype**

```
Err SrmPrimeWakeupHandler (UInt16 portId,
                            UInt16 minBytes)
```

**Parameters**

- `-> portId`  
  Port ID returned from SrmOpen or SrmExtOpen.

- `-> minBytes`  
  Number of bytes that must be received before wakeup handler is called. Typically, this is set to 1.

**Result**

This function returns the following error codes:

- **errNone**  
  No error.
serErrBadPort    This port doesn’t exist.
serErrNotOpen    The port is not open.
serErrNoDevicesAvail    No serial devices could be found.

Comments    This function primes a wakeup handler installed by SrmSetWakeupHandler.

Compatibility    Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also    SrmSetWakeupHandler, WakeupHandlerProcPtr

SrmReceive

Purpose    Receives a specified number of bytes.

Declared In    SerialMgr.h

Prototype    UInt32 SrmReceive (UInt16 portId, void *rcvBufP, UInt32 count, Int32 timeout, Err *errP)

Parameters
- portID    Port ID returned from SrmOpen or SrmExtOpen.
- rcvBufP    Pointer to buffer where received data is to be returned.
- count    Length of data buffer (in bytes). This specifies the number of bytes to receive.
- timeout    The amount of time (in ticks) that the Serial Manager waits to receive the requested block of data. At the end of the timeout, data received up to that time is returned.
- errP    Error code.

Result    Number of bytes of data actually received.
Serial Manager
Serial Manager Functions

Comments
The following error codes can be returned in errP:

errNone No error.
serErrBadPort This port doesn’t exist.
serErrNotOpen The port is not open.
serErrTimeOut Unable to receive data within the specified timeout period.
serErrConfigurationFailed The port needs time to configure, and the configuration has failed.
serErrNotSupported The port is not the foreground port.
serErrConfigurationFailed The port could not configure itself.
serErrLineErr A line error occurred during the receipt of data. Use SrmGetStatus to obtain the exact line error.
serErrNoDevicesAvail No serial devices could be found.

Compatibility
Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmReceiveCheck, SrmReceiveFlush, SrmReceiveWait
SrmReceiveCheck

Purpose  Checks the receive FIFO and returns the number of bytes in the serial receive queue.

Declared In  SerialMgr.h

Prototype  Err SrmReceiveCheck (UInt16 portId, UInt32 *numBytesP)

Parameters
- portID  Port ID returned from SrmOpen or SrmExtOpen.
- numBytesP  Number of bytes in the receive queue.

Result  This function returns the following error codes:
- errNone  No error.
- serErrBadPort  This port doesn’t exist.
- serErrNotOpen  The port is not open.
- serErrLineErr  A line error has occurred. Use SrmGetStatus to obtain the exact line error.

Compatibility  Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also  SrmReceive, SrmReceiveFlush, SrmReceiveWait

SrmReceiveFlush

Purpose  Flushes the receive FIFOs.

Declared In  SerialMgr.h

Prototype  Err SrmReceiveFlush (UInt16 portId, Int32 timeout)

Parameters
- portId  Port ID returned from SrmOpen or SrmExtOpen.
-> timeout       Timeout value, in ticks.

**Result**  
This function returns the following error codes:

- **errNone** No error.
- **serErrBadPort** This port doesn’t exist.
- **serErrNotOpen** The port is not open.
- **serErrNotSupported** The port is not the foreground port.
- **serErrNoDevicesAvail** No serial devices could be found.

**Comments**  
The `timeout` value forces this function to wait a period of ticks after flushing the port to see if more data shows up to be flushed. If more data arrives within the timeout period, the port is flushed again and the timeout counter is reset and waits again. The function only exits after no more bytes are received by the port for the full timeout period since the last flush of the port. To avoid this waiting behavior, specify 0 for the timeout period.

Any errors on the line are cleared before this function returns.

**Compatibility**  
Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

**See Also**  
[SrmReceive](#), [SrmReceiveCheck](#), [SrmReceiveWait](#)
SrmReceiveWait

Purpose  Waits until some number of bytes of data have arrived into the serial receive queue, then returns.

Declared In  SerialMgr.h

Prototype  Err SrmReceiveWait (UInt16 portId, UInt32 bytes, Int32 timeout)

Parameters
-> portID  Port ID returned from SrmOpen or SrmExtOpen.
-> bytes  Number of bytes to wait for.
-> timeout  Timeout value, in ticks.

Result  This function returns the following error codes:
errNone  No error.
serErrBadPort  This port doesn’t exist.
serErrNotOpen  The port is not open.
serErrTimeOut  Unable to receive data within the specified timeout period.
serErrNotSupported  The port is not the foreground port.
serErrBadParam  The bytes parameter exceeds the size of the receive queue. Use SrmSetReceiveBuffer to increase the size of the receive queue.
serErrLineErr  A line error occurred during the receipt of data. Use SrmGetStatus to obtain the exact line error.
serErrNoDevicesAvail  No serial devices could be found.

Comments  If this function returns no error, the application can either check the number of bytes currently in the receive queue (using
Serial Manager
Serial Manager Functions

SrmReceiveCheck) or it can just specify a buffer and receive the data by calling SrmReceive.

Do not call SrmReceiveWait from within a wakeup handler. If you do, the serErrTimeOut error is returned.

Compatibility
Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmReceive, SrmReceiveCheck, SrmReceiveFlush

SrmReceiveWindowClose

Purpose
Closes direct access to the Serial Manager’s receive queue.

Declared In
SerialMgr.h

Prototype
Err SrmReceiveWindowClose (UInt16 portId, UInt32 bytesPulled)

Parameters
-> portId Port ID returned from SrmOpen or SrmExtOpen.
-> bytesPulled Number of bytes the application read from the receive queue.

Result
This function returns the following error codes:

errNone No error.
serErrBadPort This port doesn’t exist.
serErrNotOpen The port is not open.
serErrNotSupported The port is not the foreground port.
serErrNoDevicesAvail No serial devices could be found.

Comments Call this function when the application has read as many bytes as it needs out of the receive queue or it has read all the available bytes.
Compatibility

Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also
SrmReceiveWindowOpen

SrmReceiveWindowOpen

Purpose
Provides direct access to the Serial Manager’s receive queue.

Declared In
SerialMgr.h

Prototype
Err SrmReceiveWindowOpen (UInt16 portId, UInt8 **bufPP, UInt32 *sizeP)

Parameters

- portId
  Port ID returned from SrmOpen or SrmExtOpen.

- bufPP
  Pointer to a pointer to the receive buffer.

- sizeP
  Available bytes in buffer.

Result
This function returns the following error codes:

- errNone
  No error.

- serErrBadPort
  This port doesn’t exist.

- serErrNotOpen
  The port is not open.

- serErrNotSupported
  The port is not the foreground port.

- serErrLineErr
  The data in the queue contains line errors.

- serErrNoDevicesAvailable
  No serial devices could be found.

Comments
This function lets applications directly access the Serial Manager’s receive queue to eliminate buffer copying by the Serial Manager. This access is a “back door” route to the received data. After retrieving data from the buffer, the application must call SrmReceiveWindowClose.
Applications that want to empty the receive buffer entirely should call the SrmReceiveWindowOpen and SrmReceiveWindowClose functions repeatedly until the buffer size returned is 0.

**IMPORTANT:** Once an application calls SrmReceiveWindowOpen, it should not attempt to receive data via the normal method of calling SrmReceive or SrmReceiveWait, as these functions interfere with direct access to the receive queue.

**Compatibility**

Implemented only if New Serial Manager Feature Set Version 1 is present.

**See Also**

SrmReceiveWindowClose

**SrmSend**

**Purpose**

Sends a block of data out the specified port.

**Declared In**

SerialMgr.h

**Prototype**

UInt32 SrmSend (UInt16 portId, const void *bufP, UInt32 count, Err *errP)

**Parameters**

- **-> portID**
  Port ID returned from SrmOpen or SrmExtOpen.

- **-> bufp**
  Pointer to data to send.

- **-> count**
  Length of data buffer, in bytes.

- **<- errP**
  Error code. See the Comments section for details.

**Result**

Number of bytes of data actually sent.

**Comments**

When SrmSend returns, you should check the value returned in the errP parameter. If errNone, then the entire data buffer was sent. If
not errNone, then the result equals the number of bytes sent before the error occurred. The possible error values are:

- **errNone**  
  No error.
- **serErrBadPort**  
  This port doesn’t exist.
- **serErrNotOpen**  
  The port is not open.
- **serErrTimeOut**  
  Unable to send data within the specified CTS timeout period.
- **serErrNoDevicesAvail**  
  No serial devices could be found.
- **serErrConfigurationFailed**  
  The port configuration has failed.
- **serErrNotSupported**  
  The specified port is not the foreground port.

**Compatibility**  
Implemented only if New Serial Manager Feature Set Version 1 is present.

**See Also**  
SrmSendCheck, SrmSendFlush, SrmSendWait

### SrmSendCheck

**Purpose**  
Checks the transmit FIFO and returns the number of bytes left to be sent.

**Declared In**  
SerialMgr.h

**Prototype**  
```c
Err SrmSendCheck (UInt16 portId,
                  UInt32 *numBytesP)
```

**Parameters**  
- **portID**  
  Port ID returned from SrmOpen or SrmExtOpen.
- **numBytesP**  
  Number of bytes left in the FIFO queue.

**Result**  
This function returns the following error codes:

- **errNone**  
  No error.
Serial Manager
Serial Manager Functions

serErrBadPort  This port doesn’t exist.
serErrNotOpen  The port is not open.
serErrNotSupported  This feature not supported by the hardware.
serErrNoDevicesAvail  No serial devices could be found.

Comments  Not all serial devices support this feature.

Compatibility  Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also  SrmSend, SrmSendFlush, SrmSendWait

SrmSendFlush

Purpose  Flushes the transmit FIFO.

Declared In  SerialMgr.h

Prototype  Err SrmSendFlush (UInt16 portId)

Parameters  -> portID  Port ID returned from SrmOpen or SrmExtOpen.

Result  This function returns the following error codes:
errNone  No error.
serErrBadPort  This port doesn’t exist.
serErrNotOpen  The port is not open.
serErrNotSupported  The port is not the foreground port.
serErrNoDevicesAvail  No serial devices could be found.
Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmSend, SrmSendCheck, SrmSendWait

SrmSendWait

Purpose Waits until all previous data has been sent from the transmit FIFO, then returns.

Declared In SerialMgr.h

Prototype Err SrmSendWait (UInt16 portId)

Parameters -> portID Port ID returned from SrmOpen or SrmExtOpen.

Result This function returns the following error codes:

errNone No error.
serErrBadPort This port doesn’t exist.
serErrNotOpen The port is not open.
serErrTimeOut Unable to send data within the CTS timeout period.

Comments Consider calling this function if your software needs to detect when all data has been transmitted by SrmSend. The SrmSend function blocks until all data has been transmitted or a timeout occurs. A subsequent call to SrmSendWait blocks until all data queued up for transmission has been transmitted or until another CTS timeout occurs (if CTS handshaking is enabled).
Serial Manager
Serial Manager Functions

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmSend, SrmSendCheck, SrmSendFlush

SrmSetReceiveBuffer

Purpose Installs a new buffer into the Serial Manager’s receive queue.

Declared In SerialMgr.h

Prototype Err SrmSetReceiveBuffer (UInt16 portId, void *bufP, UInt16 bufSize)

Parameters -> portID Port ID returned from SrmOpen or SrmExtOpen.
-> bufP Pointer to new receive buffer. Ignored if bufSize is NULL.
-> bufSize Size of new receive buffer in bytes. To remove this buffer and allocate a new default buffer (512 bytes), specify NULL.

Result This function returns the following error codes:
errNone No error.
serErrBadPort This port doesn’t exist.
serErrNotOpen This port is not open.
memErrNotEnoughSpace Not enough memory to allocate default buffer.
serErrNoDevicesAvail No serial devices could be found.

Comments The buffer that you pass to this function must remain allocated while you have the serial port open. Before you close the serial port, you must restore the default queue by calling SrmSetReceiveBuffer with NULL as the bufP and bufSize arguments.
IMPORTANT: Applications must install the default buffer before closing the port (or disposing of the new receive queue).

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

SrmSetWakeupHandler

Purpose Installs a wakeup handler.

Declared In SerialMgr.h

Prototype Err SrmSetWakeupHandler (UInt16 portId, WakeupHandlerProcPtr procP, UInt32 refCon)

Parameters -> portID Port ID returned from SrmOpen or SrmExtOpen.
    -> procP Pointer to a WakeupHandlerProcPtr function. Specify NULL to remove a handler.
    -> refCon User-defined data that is passed to the wakeup handler function. This can be a pointer or not.

Result This function returns the following error codes:
    errNone No error.
    serErrBadPort This port doesn’t exist.
    serErrNotOpen The port is not open.
    serErrNoDevicesAvail No serial devices could be found.

Comments The wakeup handler is a function in your application that you want to be called whenever there is data ready to be received on the specified port.

The wakeup handler function will not become active until it is primed with a number of bytes that is greater than 0, by the
SrmPrimeWakeupHandler function. Every time a wakeup handler is called, it must be re-primed (using SrmPrimeWakeupHandler) in order to be called again.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmPrimeWakeupHandler, WakeupHandlerProcPtr

Serial Manager Application-Defined Functions

WakeupHandlerProcPtr

Purpose Called after some number of bytes are received by the Serial Manager’s interrupt function.

Declared In SerialMgr.h

Prototype void (*WakeupHandlerProcPtr)(UInt32 refCon)

Parameters ->refCon User-defined data passed from the SrmSetWakeUpHandler function.

Result Returns nothing.

Comments This handler function is installed by calling SrmSetWakeUpHandler. The number of bytes after which it is called is specified by SrmPrimeWakeupHandler.

IMPORTANT: Because wakeup handlers are called during interrupt time, they cannot call any Palm OS® system functions, including SrmReceive, that may block the system in any way. Wakeup handlers should also be very short so as to reduce interrupt latency.

Two common implementations of wakeup handlers include:
• Calling EvtWakeup, which causes any pending EvtGetEvent call to return and then sends a nilEvent to the current application.

• Using SrmReceiveWindowOpen and SrmReceiveWindowClose to gain direct access to the receive queue without blocking.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmPrimeWakeupHandler, SrmSetWakeupHandler
Serial Manager
Serial Manager Application-Defined Functions
Old Serial Manager

This chapter provides reference material for the serial manager API:

- Serial Manager Data Structures
- Serial Manager Functions

The header file `SerialMgrOld.h` declares the serial manager API. For more information on the serial manager, see the chapter “Serial Communication” in the `Palm OS Programmer's Companion`, vol. II, Communications.

NOTE: The API described in this chapter is obsolete if the New Serial Manager Feature Set is present. The API is still supported for backward compatibility; however, the Serial Manager APIs are preferred.

Serial Manager Data Structures

**SerCtlEnum**

To perform a control function, applications call `SerControl`, which performs one of the control operations specified by `SerCtlEnum`, which has the following elements:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serCtlFirstReserved = 0</td>
<td>Reserve 0</td>
</tr>
<tr>
<td>serCtlStartBreak</td>
<td>Turn RS232 break signal on. Applications have to make sure that the break is set long enough to generate a value BREAK! valueP = 0; valueLenP = 0</td>
</tr>
<tr>
<td>serCtlStopBreak</td>
<td>Turn RS232 break signal off: valueP = 0; valueLenP = 0</td>
</tr>
</tbody>
</table>
The `SerSettingsType` structure defines serial port attributes; it is used by the calls `SerGetSettings` and `SerSetSettings`. The `SerSettingsPtr` type points to a `SerSettingsType` structure.

```c
typedef struct SerSettingsType {
    UInt32 baudRate;
    ...
} SerSettingsType;
```

### SerSettingsType

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serCtlBreakStatus</td>
<td>Get RS232 break signal status (on or off): valueP = ptr to Word for returning status (0 = off, !0 = on)</td>
</tr>
<tr>
<td></td>
<td>*valueLenP = sizeof(Word)</td>
</tr>
<tr>
<td>serCtlStartLocalLoopback</td>
<td>Start local loopback test; valueP = 0, valueLenP = 0</td>
</tr>
<tr>
<td>serCtlStopLocalLoopback</td>
<td>Stop local loopback test; valueP = 0, valueLenP = 0</td>
</tr>
<tr>
<td>serCtlMaxBaud</td>
<td>valueP = ptr to DWord for returned baud</td>
</tr>
<tr>
<td></td>
<td>*valueLenP = sizeof(DWord)</td>
</tr>
<tr>
<td>serCtlHandshakeThreshold</td>
<td>Retrieve HW handshake threshold; this is the maximum baud rate that does not require hardware handshaking</td>
</tr>
<tr>
<td></td>
<td>valueP = ptr to DWord for returned baud</td>
</tr>
<tr>
<td></td>
<td>*valueLenP = sizeof(DWord)</td>
</tr>
<tr>
<td>serCtlEmuSetBlockingHook</td>
<td>Set a blocking hook routine. valueP = ptr to SerCallbackEntryType</td>
</tr>
<tr>
<td></td>
<td>*valueLenP = sizeof(SerCallbackEntryType)</td>
</tr>
<tr>
<td></td>
<td>Returns the old settings in the first argument.</td>
</tr>
<tr>
<td>serCtlLAST</td>
<td>Add new address entries before this one.</td>
</tr>
</tbody>
</table>
Serial Manager Functions

SerClearErr

Purpose  Reset the serial port’s line error status.

Declared In  SerialMgrOld.h

Prototype  Err SerClearErr (UInt16 refNum)

Parameters  -> refNum     The serial library reference number.

Result  0     No error.

Comments  Call SerClearErr only after a serial manager function
           (SerReceive, SerReceiveCheck, SerSend, etc.) returns with
           the error code serErrLineErr.

           The reason for this is that SerClearErr resets the serial port. So, if
           SerClearErr is called unconditionally while a byte is coming into
           the serial port, that byte is guaranteed to become corrupted.
The right strategy is to always check the error code returned by a serial manager function. If it’s `serErrLineErr`, call `SerClearErr` immediately. However, don’t make unsolicited calls to `SerClearErr`.

When you get `serErrLineErr`, consider flushing the receive queue for a fraction of a second by calling `SerReceiveFlush`. `SerReceiveFlush` calls `SerClearErr` for you.

**SerClose**

**Purpose**  
Release the serial port previously acquired by `SerOpen`.

**Declared In**  
`SerialMgrOld.h`

**Prototype**  
```
Err SerClose (UInt16 refNum)
```

**Parameters**  
- `refNum`  
  Serial library reference number.

**Result**  
- 0  
  No error.
- `serErrNotOpen`  
  Port wasn’t open.
- `serErrStillOpen`  
  Port still held open by another process.

**Comments**  
Releases the serial port and shuts down serial port hardware if the open count has reached 0. Open serial ports consume more energy from the device’s batteries; it’s therefore essential to keep a port open only as long as necessary.

**Caveat**  
Don’t call `SerClose` unless the return value from `SerOpen` was 0 (zero) or `serErrAlreadyOpen`.

**See Also**  
`SerOpen`
SerControl

Purpose
Perform a control function.

Declared In
SerialMgrOld.h

Prototype
Err SerControl (UInt16 refNum, UInt16 op, void *valueP, UInt16 *valueLenP)

Parameters
- refNum Reference number of library.
- op Control operation to perform (SerCtlEnum).
- valueP Pointer to value for operation.
- valueLenP Pointer to size of value.

Result
0 No error.
serErrBadParam Invalid parameter (unknown).
serErrNotOpen Library not open.

Comments
This function provides extensible control features for the serial manager. You can
- Turn on/off the RS232 break signal and check its status.
- Perform a local loopback test.
- Get the maximum supported baud rate.
- Get the hardware handshake threshold baud rate.

Compatibility
Implemented only if 2.0 New Feature Set is present.
Old Serial Manager
Serial Manager Functions

SerGetSettings

**Purpose**
Fill in the `SerSettingsType` structure with current serial port attributes.

**Declared In**
SerialMgrOld.h

**Prototype**
```
Err SerGetSettings (UInt16 refNum,
                  SerSettingsPtr settingsP)
```

**Parameters**
- `refNum` Serial library reference number.
- `settingsP` Pointer to `SerSettingsType` structure to be filled in.

**Result**
- `0` No error.
- `serErrNotOpen` The port wasn’t open.

**Comments**
The information returned by this call includes the current baud rate, CTS timeout, handshaking options, and data format options.
See the `SerSettingsType` structure for more details.

**See Also**
SerSend

SerGetStatus

**Purpose**
Return the pending line error status for errors that have been detected since the last time `SerClearErr` was called.

**Declared In**
SerialMgrOld.h

**Prototype**
```
UInt16 SerGetStatus (UInt16 refNum,
                    Boolean *ctsOnP, Boolean *dsrOnP)
```

**Parameters**
- `refNum` Serial library reference number.
- `ctsOnP` Pointer to location for storing a Boolean value.
>- dsrOnP Pointer to location for storing a Boolean value.

**Result** Returns any combination of the following constants, bitwise OR’ed together:

- **serLineErrorParity**
  - Parity error.
- **serLineErrorHWOverrun**
  - Hardware overrun.
- **serLineErrorFraming**
  - Framing error.
- **serLineErrorBreak**
  - Break signal detected.
- **serLineErrorHShake**
  - Line handshake error.
- **serLineErrorSWOverrun**
  - Software overrun.

**Comments** When another serial manager function returns an error code of **serErrLineErr**, **SerGetStatus** can be used to find out the specific nature of the line error(s).

The values returned via **ctsOnP** and **dsrOnP** are not meaningful in the present version of the software.

**See Also** [SerClearErr](#)

---

**SerOpen**

**Purpose** Acquire and open a serial port with given baud rate and default settings.

**Declared In** `SerialMgrOld.h`

**Prototype**

```
Err SerOpen (UInt16 refNum, UInt16 port,
UInt32 baud)
```

**Parameters**

- `-> refNum` Serial library reference number.
Old Serial Manager
Serial Manager Functions

- port Port number.
- baud Baud rate.

**Result**
- 0 No error.
- serErrAlreadyOpen Port was open. Enables port sharing by “friendly” clients (not recommended).
- serErrBadParam Invalid parameter.
- memErrNotEnoughSpace Insufficient memory.

**Comments**
Acquires the serial port, powers it up, and prepares it for operation. To obtain the serial library reference number, call `SysLibFind` with “Serial Library” as the library name. This reference number must be passed as a parameter to all serial manager functions. The device currently contains only one serial port with port number 0 (zero).

The baud rate is an integral baud value (for example - 300, 1200, 2400, 9600, 19200, 38400, 57600, etc.). The Palm OS® device has been tested at the standard baud rates in the range of 300 - 57600 baud. Baud rates through 1 Mbit are theoretically possible. Use CTS handshaking at baud rates above 19200 (see `SerSetSettings`).

An error code of 0 (zero) or `serErrAlreadyOpen` indicates that the port was successfully opened. If the port is already open when `SerOpen` is called, the port’s open count is incremented and an error code of `serErrAlreadyOpen` is returned. This ability to open the serial port multiple times allows cooperating tasks to share the serial port. Other tasks must refrain from using the port if `serErrAlreadyOpen` is returned and close it by calling `SerClose`.

1600   *Palm OS Programmer’s API Reference*
SerReceive

Purpose  Receives size bytes worth of data or returns with error if a line error or timeout is encountered.

Declared In  SerialMgrOld.h

Prototype  

```c
UInt32 SerReceive (UInt16 refNum, void *bufP, 
UInt32 count, Int32 timeout, Err* errP)
```

Parameters  

- refNum  Serial library reference number.
- bufP  Buffer for receiving data.
- count  Number of bytes to receive.
- timeout  Interbyte timeout in ticks, 0 for none, -1 forever.
- errP  For returning error code.

Result  Number of bytes received:

- *errP = 0  No error.
- serErrLineErr  RS232 line error.
- serErrTimeOut  Interbyte timeout.

Compatibility  Implemented only if 2.0 New Feature Set is present.

NOTE:  The old versions of SerSend and SerReceive are still available as SerSend10 and SerReceive10 (not V10).

See Also  SerReceive10
SerReceive10

Purpose
Receive a stream of bytes.

Declared In
SerialMgrOld.h

Prototype
Err SerReceive10 (UInt16 refNum, void *bufP,
UInt32 bytes, Int32 timeout)

Parameters
- refNum The serial library reference number.
- bufP Pointer to the buffer for receiving data.
- bytes Number of bytes desired.
- timeout Interbyte time out in system ticks (-1 = forever).

Result
0 No error. Requested number of bytes was received.
serErrTimeOut Interbyte time out exceeded while waiting for the next byte to arrive.
serErrLineErr Line error occurred (see SerClearErr and SerGetStatus).

Comments
SerReceive blocks until all the requested data has been received or an error occurs. Because this call returns immediately without any data if line errors are pending, it is important to acknowledge the detection of line errors by calling SerClearErr. If you just need to retrieve all or some of the bytes which are already in the receive queue, call SerReceiveCheck first to get the count of bytes presently in the receive queue.

Compatibility
This function corresponds to the 1.0 version of SerReceive.
SerReceiveCheck

**Purpose**
Return the count of bytes presently in the receive queue.

**Declared In**
SerialMgrOld.h

**Prototype**
```
Err SerReceiveCheck (UInt16 refNum, 
UInt32 *numBytesP)
```

**Parameters**
- `refNum` Serial library reference number.
- `numBytesP` Pointer to location for returning the byte count.

**Result**
- `0` No error.
- `serErrLineErr` Line error pending (see SerClearErr and SerGetStatus).

**Comments**
Because this call does not return the byte count if line errors are pending, it is important to acknowledge the detection of line errors by calling SerClearErr.

**See Also**
SerReceiveWait

SerReceiveFlush

**Purpose**
Discard all data presently in the receive queue and flush bytes coming into the serial port. Clear the saved error status.

**Declared In**
SerialMgrOld.h

**Prototype**
```
void SerReceiveFlush (UInt16 refNum, 
Int32 timeout)
```

**Parameters**
- `refNum` Serial library reference number.
- `timeout` Interbyte time out in system ticks (-1 = forever).

**Result**
Returns nothing.
Old Serial Manager
Serial Manager Functions

Comments SerReceiveFlush blocks until a timeout occurs while waiting for the next byte to arrive.

SerReceiveWait

Purpose Wait for at least bytes of data to accumulate in the receive queue.

Declared In SerialMgrOld.h

Prototype Err SerReceiveWait (UInt16 refNum, UInt32 bytes, Int32 timeout)

Parameters -> refNum Serial library reference number.
-> bytes Number of bytes desired.
-> timeout Interbyte timeout in system ticks (-1 = forever).

Result 0 No error.
serErrTimeOut Interbyte timeout exceeded while waiting for next byte to arrive.
serErrLineErr Line error occurred (see SerClearErr and SerGetStatus).

Comments This is the preferred method of waiting for serial input, since it blocks the current task and allows switching the processor into a more energy-efficient state.

SerReceiveWait blocks until the desired number of bytes accumulate in the receive queue or an error occurs. The desired number of bytes must be less than the current receive queue size. The default queue size is 512 bytes. Because this call returns immediately if line errors are pending, it is important to acknowledge the detection of line errors by calling SerClearErr.

See Also SerReceiveCheck, SerSetReceiveBuffer
SerSend

Purpose
Send one or more bytes of data over the serial port.

Declared In
SerialMgrOld.h

Prototype
UInt32 SerSend (UInt16 refNum, void *bufP,
UInt32 count, Err *errP

Parameters
- refNum Serial library reference number.
- bufP Pointer to data to send.
- count Number of bytes to send.
- errP For returning error code.

Result
Returns the number of bytes transferred.
Stores in errP:
0 No error.
serErrTimeOut Handshake timeout.

The old calls worked, but they did not return enough info when they failed. The new calls (available in Palm OS v2.0 and greater) add more parameters to solve this problem and make serial communications programming simpler.

Don’t call the new functions when running on Palm OS 1.0.

Compatibility
Implemented only if 2.0 New Feature Set is present.

NOTE: The old versions of SerSend and SerReceive are still available as SerSend10 and SerReceive10 (not V10).

See Also
SerSend10, SerSendWait
SerSend10

Purpose  Send a stream of bytes to the serial port.

Declared In  SerialMgrOld.h

Prototype  Err SerSend10 (UInt16 refNum, void *bufP, UInt32 size)

Parameters
- > refNum  Serial library reference number.
- > bufP  Pointer to the data to send.
- > size  Size (in number of bytes) of the data to send.

Result
- 0  No error.
- serErrTimeOut  Handshake timeout (such as waiting for CTS to become asserted).

Comments  In the present implementation, SerSend10 blocks until all data is transferred to the UART or a timeout error (if CTS handshaking is enabled) occurs. Future implementations may queue up the request and return immediately, performing transmission in the background. If your software needs to detect when all data has been transmitted, see SerSendWait.

This routine observes the current CTS time out setting if CTS handshaking is enabled (see SerGetSettings and SerSend).

Compatibility  This function corresponds to the 1.0 version of SerSend.

See Also  SerSend, SerSendWait
SerSendFlush

Purpose    Discard all data presently in the transmit queue.

Declared In    SerialMgrOld.h

Prototype    Err SerSendFlush (UInt16 refNum)

Parameters
- refNum    Serial library reference number.

Result
- 0    No error.

See Also    SerSend, SerSendWait

SerSendWait

Purpose    Wait until the serial transmit buffer empties.

Declared In    SerialMgrOld.h

Prototype    Err SerSendWait (UInt16 refNum, Int32 timeout)

Parameters
- refNum    Serial library reference number.
- timeout    Reserved for future enhancements. Set to (-1) for compatibility.

Result
- 0    No error.
- serErrTimeOut    Handshake timeout (such as waiting for CTS to become asserted).

Comments    SerSendWait blocks until all data is transferred or a timeout error (if CTS handshaking is enabled) occurs. This routine observes the current CTS timeout setting if CTS handshaking is enabled (see SerGetSettings and SerSend).

See Also    SerSend
**SerSetReceiveBuffer**

**Purpose**
Replace the default receive queue. To restore the original buffer, pass `bufSize = 0`.

**Declared In**
SerialMgrOld.h

**Prototype**
```
Err SerSetReceiveBuffer (UInt16 refNum, void *bufP, UInt16 bufSize)
```

**Parameters**
- `refNum` Serial library reference number.
- `bufP` Pointer to buffer to be used as the new receive queue.
- `bufSize` Size of buffer, or 0 to restore the default receive queue.

**Result**
Returns 0 if successful.

**Comments**
The specified buffer needs to contain 32 extra bytes for serial manager overhead (its size should be your application's requirement plus 32 bytes). The default receive queue must be restored before the serial port is closed. To restore the default receive queue, call `SerSetReceiveBuffer` passing 0 (zero) for the buffer size. The serial manager does not free the custom receive queue.

**SerSetSettings**

**Purpose**
Set the serial port settings; that is, change its attributes.

**Declared In**
SerialMgrOld.h

**Prototype**
```
Err SerSetSettings (UInt16 refNum, SerSettingsPtr settingsP)
```

**Parameters**
- `refNum` Serial library reference number.
<-> settingsP Pointer to the filled in \texttt{SerSettingsType} structure.

\textbf{Result} 0 No error.

serErrNotOpen The port wasn’t open.
serErrBadParam Invalid parameter.

\textbf{Comments} The attributes set by this call include the current baud rate, CTS timeout, handshaking options, and data format options. See the definition of the \texttt{SerSettingsType} structure for more details.

To do 7E1 transmission, OR together:

serSettingsFlagBitsPerChar7 | serSettingsFlagParityOnM | serSettingsFlagParityEvenM | serSettingsFlagStopBits1

If you’re trying to communicate at speeds greater than 19.2 Kbps, you need to use hardware handshaking:

serSettingsFlagRTSAutoM | serSettingsFlagCTSAutoM.

\textbf{See Also} \texttt{SerGetSettings}
Serial Link Manager

This chapter provides reference material for the serial link manager API. The header file SerialLinkMgr.h declares the serial link manager API. For more information on the serial link manager, see the chapter “Serial Communication” in the Palm OS Programmer’s Companion, vol. II, Communications.

Serial Link Manager Functions

**SlkClose**

**Purpose** Close down the serial link manager.

**Declared In** SerialLinkMgr.h

**Prototype** Err SlkClose (void)

**Parameters** None.

**Result**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error.</td>
</tr>
<tr>
<td>slkErrNotOpen</td>
<td>The serial link manager was not open.</td>
</tr>
</tbody>
</table>

**Comments** When the open count reaches zero, this routine frees resources allocated by serial link manager.
Serial Link Manager
Serial Link Manager Functions

SlkCloseSocket

**Purpose**
Closes a socket previously opened with `SlkOpenSocket`. The caller is responsible for closing the communications library used by this socket, if necessary.

**Declared In**
SerialLinkMgr.h

**Prototype**
```c
Err SlkCloseSocket (UInt16 socket)
```

**Parameters**
- `socket`  
The socket ID to close.

**Result**
- 0  
No error.
- `slkErrSocketNotOpen`  
The socket was not open.

**Comments**
`SlkCloseSocket` frees system resources the serial link manager allocated for the socket. It does not free resources allocated and passed by the client, such as the buffers passed to `SlkSetSocketListener`; this is the client’s responsibility. The caller is also responsible for closing the communications library used by this socket.

**See Also**
`SlkOpenSocket`

SlkFlushSocket

**Purpose**
Flush the receive queue of the communications library associated with the given socket.

**Declared In**
SerialLinkMgr.h

**Prototype**
```c
Err SlkFlushSocket (UInt16 socket, Int32 timeout)
```

**Parameters**
- `-> socket`  
Socket ID.
Serial Link Manager
Serial Link Manager Functions

timeout
Interbyte timeout in system ticks.

Result
0
No error.

slkErrSocketNotOpen
The socket wasn’t open.

SlkOpen

Purpose
Initialize the serial link manager.

Declared In
SerialLinkMgr.h

Prototype
Err SlkOpen (void)

Parameters
None.

Result
0
No error.

slkErrAlreadyOpen
No error.

Comments
Initializes the serial link manager, allocating necessary resources. Return codes of 0 (zero) and slkErrAlreadyOpen both indicate success. Any other return code indicates failure. The slkErrAlreadyOpen function informs the client that someone else is also using the serial link manager. If the serial link manager was successfully opened by the client, the client needs to call SlkClose when it finishes using the serial link manager.
SlkOpenSocket

**Purpose**
Open a serial link socket and associate it with a communications library. The socket may be a known static socket or a dynamically assigned socket.

**Declared In**
SerialLinkMgr.h

**Prototype**
Err SlkOpenSocket (UInt16 portID, UInt16 *socketP, Boolean staticSocket)

**Parameters**
- **portID**
  Comm library reference number for socket.
- **socketP**
  Pointer to location for returning the socket ID.
- **staticSocket**
  If TRUE, *socketP contains the desired static socket number to open. If FALSE, any free socket number is assigned dynamically and opened.

**Result**
- 0 No error.
  - slkErrOutOfSockets
    No more sockets can be opened.

**Comments**
The communications library must already be initialized and opened (see *SerOpen*). When finished using the socket, the caller must call *SlkCloseSocket* to free system resources allocated for the socket. For information about well-known static socket IDs, see *The Serial Link Protocol*.
**SlkReceivePacket**

**Purpose**  
Receive and validate a packet for a particular socket or for any socket. Check for format and checksum errors.

**Declared In**  
SerialLinkMgr.h

**Prototype**  
```c
Err SlkReceivePacket (UInt16 socket,
                     Boolean andOtherSockets, SlkPktHeaderPtr headerP,
                     void* bodyP, UInt16 bodySize, Int32 timeout)
```

**Parameters**
- `socket`  
The socket ID.
- `andOtherSockets`  
If TRUE, ignore destination in packet header.
- `headerP`  
Pointer to the packet header buffer (size of SlkPktHeaderType).
- `bodyP`  
Pointer to the packet client data buffer.
- `bodySize`  
Size of the client data buffer (maximum client data size which can be accommodated).
- `timeout`  
Maximum number of system ticks to wait for beginning of a packet; -1 means wait forever.

**Result**
- 0  
No error.
- slkErrSocketNotOpen  
The socket was not open.
- slkErrTimeOut  
Timed out waiting for a packet.
- slkErrWrongDestSocket  
The packet being received had an unexpected destination.
- slkErrChecksum  
Invalid header checksum or packet CRC-16.
- slkErrBuffer  
Client data buffer was too small for packet’s client data.

If `andOtherSockets` is FALSE, this routine returns with an error code unless it gets a packet for the specific socket.
If `andOtherSockets` is `TRUE`, this routine returns successfully if it sees any incoming packet from the communications library used by socket.

**Comments**
You may request to receive a packet for the passed socket ID only, or for any open socket which does not have a socket listener. The parameters also specify buffers for the packet header and client data, and a timeout. The timeout indicates how long the receiver should wait for a packet to begin arriving before timing out. If a packet is received for a socket with a registered socket listener, it will be dispatched via its socket listener procedure. On success, the packet header buffer and packet client data buffer is filled in with the actual size of the packet’s client data in the packet header’s `bodySize` field.

### SlkSendPacket

**Purpose**
Send a serial link packet via the serial output driver.

**Declared In**
`SerialLinkMgr.h`

**Prototype**
```c
Err SlkSendPacket (SlkPktHeaderPtr headerP, SlkWriteDataPtr writeList)
```

**Parameters**
- `<-> headerP`  Pointer to the packet header structure with client information filled in (see Comments).
- `-> writeList`  List of packet client data blocks (see Comments).

**Result**
- `0`  No error.
- `slkErrSocketNotOpen`  The socket was not open.
- `slkErrTimeOut`  Handshake timeout.

**Comments**
`SlkSendPacket` stuffs the signature, client data size, and the checksum fields of the packet header. The caller must fill in all other packet header fields. If the transaction ID field is set to 0 (zero), the
serial link manager automatically generates and stuffs a new non-zero transaction ID. The array of SlkWriteDataType structures enables the caller to specify the client data part of the packet as a list of noncontiguous blocks. The end of list is indicated by an array element with the size field set to 0 (zero). This call blocks until the entire packet is sent out or until an error occurs.

**SlkSetSocketListener**

**Purpose**
Register a socket listener for a particular socket.

**Declared In**
SerialLinkMgr.h

**Prototype**
Err SlkSetSocketListener (UInt16 socket, SlkSocketListenPtr socketP)

**Parameters**
- socket Socket ID.
- socketP Pointer to a SlkSocketListenType structure.

**Result**
- 0 No error.
- slkErrBadParam Invalid parameter.
- slkErrSocketNotOpen The socket was not open.

**Comments**
Called by applications to set up a socket listener.
Since the serial link manager does not make a copy of the SlkSocketListenType structure, but instead saves the passed pointer to it, the structure
- must **not** be an automatic variable (that is, local variable allocated on the stack)
- may be a global variable in an application
- may be a locked chunk allocated from the dynamic heap
The SlkSocketListenType structure specifies pointers to the socket listener procedure and the data buffers for dispatching packets destined for this socket. Pointers to two buffers must be
specified: the packet header buffer (size of SlkPktHeaderType), and the packet body (client data) buffer. The packet body buffer must be large enough for the largest expected client data size. Both buffers may be application global variables or locked chunks allocated from the dynamic heap.

The socket listener procedure is called when a valid packet is received for the socket. Pointers to the packet header buffer and the packet body buffer are passed as parameters to the socket listener procedure.

**NOTE:** The application is responsible for freeing the SlkSocketListenType structure or the allocated buffers when the socket is closed. The serial link manager doesn’t do it.

---

**Compatibility** If 5.0 New Feature Set is present this function is unimplemented.

### SlkSocketPortID

**Purpose** Get the port ID associated with a particular socket; for use with the new serial manager.

**Declared In** SerialLinkMgr.h

**Prototype**

```
ErrSlkSocketPortID (UInt16 socket, UInt16 * portIDP)
```

**Parameters**

- `-> socket` The socket ID.
- `<- portIDP` Pointer to location for returning the port ID.

**Result**

- `0` No error.
- `slkErrSocketNotOpen` The socket was not open.

**Compatibility** Implemented only if New Serial Manager Feature Set is present.
**SlkSocketSetTimeout**

**Purpose**: Set the interbyte packet receive-timeout for a particular socket.

**Declared In**: SerialLinkMgr.h

**Prototype**: 
```c
Err SlkSocketSetTimeout (UInt16 socket, Int32 timeout)
```

**Parameters**
- `socket` : Socket ID.
- `timeout` : Interbyte packet receive-timeout in system ticks.

**Result**
- 0 : No error.
- `slkErrSocketNotOpen` : The socket was not open.
Serial Link Manager
Serial Link Manager Functions
Telephony Basic Services

This chapter provides reference material for the Telephony API, which you can use to interface with telephone systems and equipment. This chapter discusses the following topics:

- **Telephony Data Structures**
- **Telephony Constants**
- **Telephony Functions**
- **Feature Support Functions**

The header file `TelephonyMgr.h` declares the telephony manager API. The header file `TelephonyMgrType.h` declares the data structures that you use with the telephony manager API.

For more information about using the telephony manager, see Chapter 10, “Telephony Manager,” in *Palm OS Programmer’s Companion*, vol. II, Communications.

Telephony Service Types

The telephony API organizes functions within sets called service sets. Each service set contains a related set of functions that may or may not be available on a particular mobile device or network. You can use the `TelIs<ServiceSet>Available` function to determine if a service set is supported in the current environment, and you can use the `TelIs<FunctionName>Supported` to determine if a specific function is supported in the current environment.
The telephony API documentation has been split into several chapters. Each chapter covers one or more of the service sets, as shown in Table 68.1

Table 68.1 Telephony service types

<table>
<thead>
<tr>
<th>Service prefix</th>
<th>Functionality</th>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tel</td>
<td>Basic</td>
<td>Chapter 68, “Telephony Basic Services.”</td>
<td>Basic functions that are always available.</td>
</tr>
<tr>
<td>TelCfg</td>
<td>Configuration</td>
<td>Chapter 69, “Telephony Security and Configuration.”</td>
<td>Allows configuration of the phone, including Short Message Services (SMS) configuration.</td>
</tr>
<tr>
<td>TelDtc</td>
<td>Data calls</td>
<td>Chapter 71, “Telephony Calls.”</td>
<td>Functions for handling data calls.</td>
</tr>
<tr>
<td>TelEmc</td>
<td>Emergency calls</td>
<td>Chapter 71, “Telephony Calls.”</td>
<td>Functions for handling emergency calls.</td>
</tr>
<tr>
<td>TelInf</td>
<td>Information</td>
<td>Chapter 68, “Telephony Basic Services.”</td>
<td>Functions for retrieving information about the phone.</td>
</tr>
<tr>
<td>TelNwk</td>
<td>Network</td>
<td>Chapter 70, “Telephony Network.”</td>
<td>Provides network oriented services, including authorization, signal level, search mode, and related operations.</td>
</tr>
<tr>
<td>TelOem</td>
<td>OEM</td>
<td>Chapter 68, “Telephony Basic Services.”</td>
<td>Provides OEMs with the ability to incorporate custom functionality.</td>
</tr>
<tr>
<td>TelPhb</td>
<td>Phone book</td>
<td>Chapter 73, “Telephony Phone Book.”</td>
<td>Functions for managing the phone book.</td>
</tr>
</tbody>
</table>
Table 68.1 Telephony service types (continued)

<table>
<thead>
<tr>
<th>Service prefix</th>
<th>Functionality</th>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TelPow</td>
<td>Power</td>
<td>Chapter 68, “Telephony Basic Services.”</td>
<td>Provides access to power supply level.</td>
</tr>
<tr>
<td>TelSms</td>
<td>Short Message Service</td>
<td>Chapter 72, “Telephony SMS.”</td>
<td>Addresses the SMS, including functions for reading, sending, replying to, and deleting short messages.</td>
</tr>
<tr>
<td>TelSnd</td>
<td>Sound</td>
<td>Chapter 68, “Telephony Basic Services.”</td>
<td>Functions for playing key tones on and muting the phone.</td>
</tr>
<tr>
<td>TelSpc</td>
<td>Speech calls</td>
<td>Chapter 71, “Telephony Calls.”</td>
<td>Function for handling voice calls, including dual tone modulated frequency (DTMF) sounds.</td>
</tr>
<tr>
<td>TelSty</td>
<td>Security</td>
<td>Chapter 69, “Telephony Security and Configuration.”</td>
<td>Supports PIN code management for access to phone and Subscriber Identity Module (SIM) security-related features.</td>
</tr>
</tbody>
</table>

Telephony Data Structures

This section describes the data structures used with the basic services portion of the telephony API.

TelEventType

The TelGetEvent and TelGetTelephonyEvent functions both return a TelEventType structure to provide information about a telephony-related event.

You call the TelGetEvent function to retrieve telephony and other events.
You call the `TelGetTelephonyEvent` function to retrieve only telephony events. This function does not consume non-telephony events.

```
typedef struct _TelEventType
{
    eventsEnum   eType;
    Boolean      penDown;
    UInt8        tapCount;
    Int16        screenX;
    Int16        screenY;
    UInt16       functionId;
    UInt16       transId;
    MemPtr       *paramP;
    Err          returnCode;
} TelEventType
```

### Field Descriptions

- **eType**: One of the `eventsEnum` constants. Specifies the type of the event.

- **penDown**: `true` if the pen was down at the time of the event, and `false` if the pen was up.

Note that this field is not filled in for telephony events.

- **tapCount**: The number of taps received at this location. This value is used mainly by text fields. When the user taps in a text field, two taps selects a word, and three taps selects the entire line.

Note that this field is not filled in for telephony events.
### Telephony Basic Services
#### Telephony Data Structures

**TelEventType**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>screenX</td>
<td>Window-relative position of the pen in pixels (number of pixels from the left bound of the window).</td>
</tr>
<tr>
<td></td>
<td>Note that this field is not filled in for telephony events.</td>
</tr>
<tr>
<td>screenY</td>
<td>Window-relative position of the pen in pixels (number of pixels from the top left of the window).</td>
</tr>
<tr>
<td></td>
<td>Note that this field is not filled in for telephony events.</td>
</tr>
<tr>
<td>functionId</td>
<td>The ID of the message associated with the function call, which specifies the telephony manager function that generated this event.</td>
</tr>
<tr>
<td>transId</td>
<td>The transaction ID that was associated with this asynchronous function call.</td>
</tr>
<tr>
<td>paramP</td>
<td>A pointer to a parameter structure that was passed when an asynchronous call was made.</td>
</tr>
<tr>
<td>returnCode</td>
<td>The return code of the asynchronously called function. The value of this field is errNone if the function succeeded, or an error code if the function failed.</td>
</tr>
</tbody>
</table>

Note that the first five fields of the TelEventType structure are the same as the first five fields of the EventType structure, which is described in Chapter 2, “Palm OS Events.”

**TelCallStateType**

The TelGetCallState function uses the TelCallState structure to retrieve information about the current state of the connected phone.

```c
typedef struct _TelGetCallStateType
```

```c
{ }
```

Palm OS Programmer’s API Reference 1625
Telephony Basic Services
Telephony Data Structures

```
UInt8    state;
UInt8    callType;
UInt8    callServiceType;
UInt8    numberSize;
Char     *number;
} TelGetCallStateType
```

**Field Descriptions**

< - state  
Filled in with one of the [Telephone Call State Constants](#), which indicates the current state of the telephone call.

< - callType  
Filled in with one of the [Telephone Call Type Constants](#), which indicates the call type of the current telephone call.

< - callServiceType  
Filled in with one of the [Telephone Call Service Type Constants](#), which indicates the call service type of the current telephone call.
TelInfGetInformationType

The TelInfGetInformation function uses the TelInfGetInformationType structure to retrieve information about the phone with which you are communicating.

typedef struct _TelInfGetInformationType
{
    UInt8 infoType;
    UInt8 size;
    UInt8 *value;
} TelInfGetInformationType

numberSize

The size of the number string buffer. When the structure is used as an input parameter, this is the allocated size, in bytes, of the buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the number buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the data, and the function using this structure generates a telErrBufferSize error.

number

A buffer into which the telephone number string is stored.

When the structure describes an incoming telephone call, this is the incoming telephone number. When the structure describes an outgoing telephone call, this is the telephone number that has been called.

Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a telErrBufferSize error.
### Field Descriptions

#### TelSendCommandStringType

The TelSendCommandString function uses the TelSendCommandStringType structure to send a command string.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>infoType</td>
<td>The type of information to retrieve. This must be one of the Information Type Constants.</td>
</tr>
<tr>
<td>size</td>
<td>The size of the value buffer. When the structure is used as an input parameter, this is the allocated size, in bytes, of the buffer. Upon return, this is the actual size of the buffer. If the value buffer is too small to contain all of the retrieved information, this field is assigned the entire length of the data, and the function using this structure generates a telErrBufferSize error.</td>
</tr>
<tr>
<td>value</td>
<td>A buffer into which the information is stored.</td>
</tr>
<tr>
<td>OemID</td>
<td>The unique ID of the OEM function set.</td>
</tr>
<tr>
<td>funcID</td>
<td>The ID of the function within the OEM function set.</td>
</tr>
<tr>
<td>paramP</td>
<td>A pointer to a parameter block that is passed to the OEM function. The OEM function might modify some of the fields in the parameter block.</td>
</tr>
</tbody>
</table>
typedef struct _TelSendCommandStringType
{
    Char    *commandString;
    Char    *resultString;
    UInt16  resultSize;
    UInt32  timeOut;
} TelSendCommandStringType

Field Descriptions

-> commandString  The command string to send.

<- resultString   The result string.

<-> resultSize     The size of the resultString string buffer. When the structure is used as an input parameter, this is the allocated size, in bytes, of the buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the resultString buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the data, and the function using this structure generates a telErrBufferSize error.

-> timeOut        The number of milliseconds before timing out.

TelSndPlayKeyToneType

The TelSndPlayKeyTone function uses the TelSndPlayKeyToneType structure to specify a key tone.

typedef struct _TelSndPlayKeyToneType
{
    UInt8    keycode;
    UInt8    type;
} TelSndPlayKeyToneType
Telephony Basic Services

Telephony Constants

Field Descriptions

- > keycode
  The keycode of the key tone to play. This must be one of the Keycode Constants.

- > type
  The tone type. This must be one of the Key Sound Type Constants.

Telephony Constants

This section describes the data structures used with the basic services portion of the telephony API, which include the following constant types:

- Battery Status Constants
- Telephone Call State Constants
- Telephone Call Type Constants
- Telephone Call Service Type Constants
- Error Code Constants
- Information Type Constants
- Keycode Constants
- Key Sound Type Constants
- Message Identifier Constants
- Service Set Constants

Battery Status Constants

The battery status constants provide information about the phone’s battery.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTelPowBatteryPowered</td>
<td>0</td>
<td>The phone is powered by a battery.</td>
</tr>
<tr>
<td>kTelPowBatteryNotPowered</td>
<td>1</td>
<td>The phone has a battery connected to it, but is not using that battery.</td>
</tr>
</tbody>
</table>
### Telephone Call State Constants

The `TelCallStateType` structure uses the telephone call state constants to encode the current state of the connected telephone call.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTelCallIdle</td>
<td>0x00</td>
<td>The connection is idle.</td>
</tr>
<tr>
<td>kTelCallConnecting</td>
<td>0x01</td>
<td>A telephone call is currently connecting.</td>
</tr>
<tr>
<td>kTelCallConnected</td>
<td>0x02</td>
<td>A telephone call is currently connected.</td>
</tr>
<tr>
<td>kTelCallRedial</td>
<td>0x03</td>
<td>A telephone call is being re-dialed.</td>
</tr>
<tr>
<td>kTelCallIncoming</td>
<td>0x04</td>
<td>A telephone call is currently incoming.</td>
</tr>
<tr>
<td>kTelCallIncomingAck</td>
<td>0x05</td>
<td>An incoming telephone call is currently being acknowledged.</td>
</tr>
<tr>
<td>kTelCallDisconnecting</td>
<td>0x06</td>
<td>A telephone call is being disconnected.</td>
</tr>
</tbody>
</table>

### Telephone Call Type Constants

The `TelCallStateType` structure uses the telephone call type constants to encode the type of the current telephone call.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTelCallTypeOutgoing</td>
<td>0x00</td>
<td>An outgoing telephone call.</td>
</tr>
<tr>
<td>kTelCallTypeIncoming</td>
<td>0x01</td>
<td>An incoming telephone call.</td>
</tr>
</tbody>
</table>
Telephone Call Service Type Constants

The `TelCallStateType` structure uses the telephone call service type constants to encode the service type of the current telephone call.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTelCallServiceVoice</td>
<td>0x00</td>
<td>A voice telephone call.</td>
</tr>
<tr>
<td>kTelCallServiceData</td>
<td>0x01</td>
<td>A data telephone call.</td>
</tr>
</tbody>
</table>

Error Code Constants

The telephony manager functions return the error code constants shown in the following table to indicate their status.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>telErrBufferSize</td>
<td>One of the buffers used to retrieve data is too small.</td>
</tr>
<tr>
<td>telErrCodingScheme</td>
<td>The coding scheme specified for the short message is not valid.</td>
</tr>
<tr>
<td>telErrCommandFailed</td>
<td>The specified command could not be performed by the phone. Check the phone driver.</td>
</tr>
<tr>
<td>telErrEntryNotFound</td>
<td>The specified entry was not found.</td>
</tr>
<tr>
<td>telErrFeatureNotSupported</td>
<td>The specified feature is not supported by the phone or network.</td>
</tr>
<tr>
<td>telErrGenericDrvNotFound</td>
<td>The generic driver could not be found.</td>
</tr>
<tr>
<td>telErrInvalidAppId</td>
<td>The specified application ID is not valid.</td>
</tr>
<tr>
<td>telErrInvalidDial</td>
<td>The dial string contains an invalid character.</td>
</tr>
<tr>
<td>telErrInvalidIndex</td>
<td>The index specified for accessing a value in storage is incorrect.</td>
</tr>
<tr>
<td>telErrInvalidParameter</td>
<td>A parameter is not valid.</td>
</tr>
<tr>
<td>telErrInvalidString</td>
<td>The text string contains an invalid character.</td>
</tr>
<tr>
<td>Constant</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>telErrLibStillInUse</td>
<td>The shared lib is currently being used by another application. Do not unload it!</td>
</tr>
<tr>
<td>telErrMemAllocation</td>
<td>A memory allocation error occurred.</td>
</tr>
<tr>
<td>telErrMsgAllocation</td>
<td>The telephony messages pool is empty; a message could not be allocated.</td>
</tr>
<tr>
<td>telErrNetworkTimeOut</td>
<td>The network did not reply within the standard time delay amount.</td>
</tr>
<tr>
<td>telErrNoNetwork</td>
<td>There is no network available.</td>
</tr>
<tr>
<td>telErrNoSIMInserted</td>
<td>The SIM card is not inserted.</td>
</tr>
<tr>
<td>telErrNoSpecificDrv</td>
<td>A specific driver was not specified.</td>
</tr>
<tr>
<td>telErrNotInstalled</td>
<td>The shared library could not be installed.</td>
</tr>
<tr>
<td>telErrPassword</td>
<td>The password is not correct.</td>
</tr>
<tr>
<td>telErrPhoneCodeRequired</td>
<td>A phone code is required.</td>
</tr>
<tr>
<td>telErrPhoneComm</td>
<td>The communication link with the phone is down.</td>
</tr>
<tr>
<td>telErrPhoneMemAllocation</td>
<td>The phone’s memory is full.</td>
</tr>
<tr>
<td>telErrPhoneMemFailure</td>
<td>The phone encountered a memory error.</td>
</tr>
<tr>
<td>telErrPhoneNumber</td>
<td>One of the following errors has occurred: the phone number is wrong, the SMS center is not valid, or the receiver phone number is wrong for the SMS.</td>
</tr>
<tr>
<td>telErrPhoneReply</td>
<td>The phone reply syntax is incorrect. Check the phone driver.</td>
</tr>
<tr>
<td>telErrPhoneToSIMPINRequired</td>
<td>A phone 2 SIM PIN code is required</td>
</tr>
<tr>
<td>telErrPIN2Required</td>
<td>A PIN2 code is required.</td>
</tr>
<tr>
<td>telErrPINRequired</td>
<td>A PIN code is required.</td>
</tr>
<tr>
<td>telErrPUK2Required</td>
<td>A PUK2 code is required.</td>
</tr>
</tbody>
</table>
## Telephony Basic Services

### Telephony Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>telErrPUKRequired</td>
<td>A PUK code is required.</td>
</tr>
<tr>
<td>telErrResultBusyResource</td>
<td>A resource is busy.</td>
</tr>
<tr>
<td>telErrResultTimeout</td>
<td>A time-out was reached.</td>
</tr>
<tr>
<td>telErrResultUserCancel</td>
<td>The user cancelled the action.</td>
</tr>
<tr>
<td>telErrSecurity</td>
<td>Access to the phone was not granted.</td>
</tr>
<tr>
<td>telErrSettings</td>
<td>The telephony settings are not valid; this is due to 1) the Phone Panel preferences do not exist, or 2) the Telephony Profile is not correctly set.</td>
</tr>
<tr>
<td>telErrSIMBusy</td>
<td>The SIM could not reply.</td>
</tr>
<tr>
<td>telErrSIMFailure</td>
<td>The SIM is not working properly.</td>
</tr>
<tr>
<td>telErrSIMWrong</td>
<td>The SIM is not accepted by the phone.</td>
</tr>
<tr>
<td>telErrSpcCallError</td>
<td>The voice telephone call encountered an error.</td>
</tr>
<tr>
<td>telErrSpcLineIsBusy</td>
<td>The voice telephone call failed.</td>
</tr>
<tr>
<td>telErrSpcLineIsReleased</td>
<td>The voice telephone call has been released.</td>
</tr>
<tr>
<td>telErrSpecificDrvNotFound</td>
<td>The specified driver could not be found.</td>
</tr>
<tr>
<td>telErrTooManyApps</td>
<td>The applications table is full.</td>
</tr>
<tr>
<td>telErrTTaskNotFound</td>
<td>The Telephony Task could not be found.</td>
</tr>
<tr>
<td>telErrTTaskNotRunning</td>
<td>The Telephony Task is not running.</td>
</tr>
<tr>
<td>telErrUnavailableValue</td>
<td>The requested value can not be retrieved at the specified time. This is usually due to a TelSpcGetCallerNumber request when there is no active line.</td>
</tr>
<tr>
<td>telErrUnknown</td>
<td>An unknown telephony manager error occurred.</td>
</tr>
<tr>
<td>telErrValidityPeriod</td>
<td>The validity period specified for the short message is not valid.</td>
</tr>
</tbody>
</table>
The `TelInfGetformationType` structure uses the information type constants to encode the type of information to retrieve about the phone.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTelInfPhoneBrand</td>
<td>0</td>
<td>The brand name of the phone.</td>
</tr>
<tr>
<td>kTelInfPhoneModel</td>
<td>1</td>
<td>The model number of the phone.</td>
</tr>
<tr>
<td>kTelInfPhoneRevision</td>
<td>2</td>
<td>The revision number of the phone.</td>
</tr>
</tbody>
</table>

**Keycode Constants**

The `TelSndPlayKeyToneType` structure uses the keycode constants to specify the key tone to play.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTe10Key</td>
<td>0x00</td>
<td>The 0 key on the phone keypad.</td>
</tr>
<tr>
<td>kTe11Key</td>
<td>0x01</td>
<td>The 1 key on the phone keypad.</td>
</tr>
<tr>
<td>kTe12Key</td>
<td>0x02</td>
<td>The 2 key on the phone keypad.</td>
</tr>
<tr>
<td>kTe13Key</td>
<td>0x03</td>
<td>The 3 key on the phone keypad.</td>
</tr>
<tr>
<td>kTe14Key</td>
<td>0x04</td>
<td>The 4 key on the phone keypad.</td>
</tr>
<tr>
<td>kTe15Key</td>
<td>0x05</td>
<td>The 5 key on the phone keypad.</td>
</tr>
<tr>
<td>kTe16Key</td>
<td>0x06</td>
<td>The 6 key on the phone keypad.</td>
</tr>
<tr>
<td>kTe17Key</td>
<td>0x07</td>
<td>The 7 key on the phone keypad.</td>
</tr>
</tbody>
</table>
Telephony Basic Services
Telephony Constants

The `TelSndPlayKeyToneType` structure uses the key sound type constants to specify how the tone is played.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTel8Key</td>
<td>0x08</td>
<td>The 8 key on the phone keypad.</td>
</tr>
<tr>
<td>kTel9Key</td>
<td>0x09</td>
<td>The 9 key on the phone keypad.</td>
</tr>
<tr>
<td>kTelPoundKey</td>
<td>0x23</td>
<td>The POUND(#) key on the phone keypad.</td>
</tr>
<tr>
<td>kTelStarKey</td>
<td>0x2A</td>
<td>The STAR(*) key on the phone keypad.</td>
</tr>
<tr>
<td>kTelSendKey</td>
<td>0x45</td>
<td>The SEND key on the phone keypad.</td>
</tr>
<tr>
<td>kTelEndKey</td>
<td>0x46</td>
<td>The END key on the phone keypad.</td>
</tr>
<tr>
<td>kTelClrKey</td>
<td>0x47</td>
<td>The CLEAR key on the phone keypad.</td>
</tr>
<tr>
<td>kTelSaveKey</td>
<td>0x48</td>
<td>The SAVE key on the phone keypad.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTelSndSingleTone</td>
<td>0x00</td>
<td>Play the key sound as a single tone.</td>
</tr>
<tr>
<td>kTelSndMultiTones</td>
<td>0x01</td>
<td>Play the key sound as a multiple tones.</td>
</tr>
</tbody>
</table>

Message Identifier Constants

The message identifier constants are used with asynchronous calls to identify which telephony function is being or has been called. The `TelMessages` enumeration defines a constant for each function name.

Each message identifier constant has the form:

```
kfunctionNameMessage
```

where `functionName` is replaced by a function name.
The following table shows examples of message identifier constants. For a complete list, see the TelephonyMgr.h file.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTelGetCallStateMessage</td>
<td>TelGetCallState</td>
</tr>
<tr>
<td>kTelNwkSelectNetworkMessage</td>
<td>TelNwkSelectNetwork</td>
</tr>
<tr>
<td>kTelSmsReadMessageMessage</td>
<td>TelSmsReadMessage</td>
</tr>
</tbody>
</table>

**Service Set Constants**

The service set constants specify a set of API services.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTelNwkServiceId</td>
<td>0</td>
<td>The network service set.</td>
</tr>
<tr>
<td>kTelStyServiceId</td>
<td>1</td>
<td>The security service set.</td>
</tr>
<tr>
<td>kTelPowServiceId</td>
<td>2</td>
<td>The power service set.</td>
</tr>
<tr>
<td>kTelCfgServiceId</td>
<td>3</td>
<td>The configuration service set.</td>
</tr>
<tr>
<td>kTelSmsServiceId</td>
<td>4</td>
<td>The short message service set.</td>
</tr>
<tr>
<td>kTelEmcServiceId</td>
<td>5</td>
<td>The emergency telephone call service set.</td>
</tr>
<tr>
<td>kTelSpcServiceId</td>
<td>6</td>
<td>The speech telephone call service set.</td>
</tr>
<tr>
<td>kTelDtcServiceId</td>
<td>7</td>
<td>The data telephone call service set.</td>
</tr>
<tr>
<td>kTelPhbServiceId</td>
<td>8</td>
<td>The phone book service set.</td>
</tr>
<tr>
<td>kTelOemServiceId</td>
<td>9</td>
<td>The OEM service set.</td>
</tr>
<tr>
<td>kTelSndServiceId</td>
<td>10</td>
<td>The sound service set.</td>
</tr>
<tr>
<td>kTelInfServiceId</td>
<td>11</td>
<td>The information service set.</td>
</tr>
</tbody>
</table>

**Telephony Functions**

This section describes the functions used with the basic services portion of the telephony API.
**TelCancel**

**Purpose**
Cancels an asynchronous function call.

**Declared In**
TelephonyMgr.h

**Prototype**
```c
Err TelCancel(UInt16 iRefnum, TelAppID iAppId, UInt16 iTransId, UInt16 *ioTransIdP)
```

**Parameters**
- `iRefnum` The telephony manager library reference number.
- `iAppId` The telephone application attachment identifier for your application.
- `iTransId` The transaction ID associated with the function that you are cancelling.
- `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns `errNone` if the function call was successfully cancelled. Returns the `telErrCommandFailed` error code if the function call could not be cancelled.

**Asynchronous Result**
The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode` `errNone` upon success or an error code upon failure.
- `transId` The transaction ID of the operation.
- `functionId` `kTelUrqCancelMessage`
- `paramId` Points to the unsigned integer value passed to this function in the `iTransId` parameter.
Comments
This function cancels a pending asynchronous function call. You can cancel any asynchronous call except for an asynchronous call to the TelCancel function.

The function call that is cancelled returns the telErrUserCancel error code.

Compatibility
Implemented only if 4.0 New Feature Set is present.

TelClose

Purpose
Close the shared library.

Declared In
TelephonyMgr.h

Prototype
Err TelClose(UInt16 iRefnum, TelAppID iAppId)

Parameters
- iRefnum
  The telephony manager library reference number.
- iAppId
  The telephone application attachment identifier for your application.

Result
Returns an error code, or error none if the library was successfully closed. If the library is currently being used by another application, this function returns the telErrLibStillInUse error code.

Comments
Call this function when you are done with the telephony manager. You can only use this function synchronously.

If no other application is using the telephony manager, this function stops the Telephony task and releases any resources used by the telephony manager.

See Also
TelOpen
**TelClosePhoneConnection**

**Purpose**  Closes down communications with the connected phone.

**Declared In**  TelephonyMgr.h

**Prototype**  
```c
Err TelClosePhoneConnection(UInt16 iRefnum, TelAppID iAppId, UInt16 *ioTransIdP)
```

**Parameters**

- `iRefnum`  The telephony manager library reference number.
- `iAppId`  The telephone application attachment identifier for your application.
- `ioTransIdP`  Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**  Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**  The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode`  `errNone` upon success or an error code upon failure.
- `transId`  The transaction ID of the operation.
- `paramP`  A NULL pointer.
- `functionId`  `kTelUrqClosePhoneConnectionMessage`

**Comments**  Call this function when you have finished communications with the phone and are ready to disconnect from it.
**Compatibility**  Implemented only if 4.0 New Feature Set is present.

**See Also**  TelOpenPhoneConnection

---

### TelGetCallState

**Purpose**  Retrieves the current telephone call state information.

**Declared In**  TelephonyMgr.h

**Prototype**  
```c
Err TelGetCallState(UInt16 iRefnum,
TelAppID iAppId, TelGetCallStateType *ioParamP,
UInt16 *ioTransIdP)
```

**Parameters**

- **-> iRefnum**  The telephony manager library reference number.

- **-> iAppId**  The telephone application attachment identifier for your application.

- **<-> ioParamP**  A pointer to a `TelCallStateType` structure that describes the state of the current telephone call.

  On input, the `numberSize` field of this structure specifies the allocated size of the number buffer. Upon return, the `numberSize` field specifies the actual size of the telephone number, even if it was truncated to fit into the buffer.

- **<-> ioTransIdP**  Set the value of this parameter to NULL to cause the function to execute synchronously.

  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**  Returns `errNone` if the function was successful or returns an error code if not successful.
Asynchronous Result

The following fields are updated in the **TelEventType** event that is sent when the operation completes:

- `returnCode`  
  errNone upon success or an error code upon failure.

- `transId`  
  The transaction ID of the operation.

- `paramP`  
  Points to the **TelCallStateType** structure passed to this function in the `ioCallState` parameter.

- `functionId`  
  kTelGetCallStateMessage

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

Comments

This function retrieves information about the current telephone call state of the connection with the phone, and stores that information into the supplied **TelCallStateType** structure.

The current incoming or outgoing telephone call number is stored into the `number` field of the **TelCallStateType** structure referenced by `ioCallStateP`. If the `number` field buffer is too small to contain the complete telephone number, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The `numberSize` field of the structure is always updated to contain the actual size of the complete telephone number.

Compatibility

Implemented only if **4.0 New Feature Set** is present.

See Also  
**TelSpcAcceptCall, TelSpcCallNumber**
**TelGetEvent**

**Purpose**
Retrieves events for applications that use the telephony manager.

**Declared In**
TelephonyMgr.h

**Prototype**
void TelGetEvent(UInt16 iRefnum, TelAppID iAppId, EventPtr oEventP, Int32 iTimeout)

**Parameters**
- `iRefnum` The telephony manager library reference number.
- `iAppId` The telephone application attachment identifier for your application.
- `oEventP` A pointer to a TelEventType structure. Upon return, the structure contains the event information, which you should use as described in the Comments section.
- `iTimeout` Maximum number of ticks to wait before an event is returned (`evtWaitForever` means wait indefinitely).

**Result**
Returns nothing.

**Comments**
This function retrieves both telephony and standard Palm OS® events. You must call this function to retrieve events in any application that is running in the UI task and using the telephony manager.

Upon return from this function, you need to test the type of the event by examining the `oEventP->type` field. If the event type is a telephony event, then you need to cast the pointer as follows to access the fields:

```c
TelEventType *telEventP = (TelEventType *)oEventP;
```

This function calls both the EvtGetEvent and TelGetTelephonyEvent functions to retrieve the next event for your application.
Telephony Basic Services
Telephony Functions

TelGetTelephonyEvent

Purpose
Retrieves telephony events only.

Declared In
TelephonyMgr.h

Prototype
void TelGetTelephonyEvent(UInt16 iRefnum, TelAppID iAppId, EventPtr oEventP, Int32 iTimeOut)

Parameters
-> iRefnum The telephony manager library reference number.

-> iAppId The telephone application attachment identifier for your application.

<- oEventP A pointer to a TelEventType structure. Upon return, the structure contains the event information.

-> iTimeout Maximum number of ticks to wait before an event is returned (evtWaitForever means wait indefinitely).

Result
Returns nothing.

Comments
Use this function instead of the TelGetEvent function when you only want to process telephony events.

Compatibility
Implemented only if 4.0 New Feature Set is present.

See Also
EvtGetEvent, TelGetTelephonyEvent
TellInfGetInformation

**Purpose**  Retrieve brand, model, and revision information for the phone.

**Declared In**  TelephonyMgr.h

**Prototype**  
```c
Err TellInfGetInformation(UInt16 iRefnum,
TelAppID iAppId,
TellInfGetInformationType *ioParamP,
UInt16 *ioTransIdP)
```

**Parameters**

- **iRefnum**  The telephony manager library reference number.
- **iAppId**  The telephone application attachment identifier for your application.
- **ioParamP**  A pointer to a `TellInfGetInformationType` structure. On input, the `infoType` field of the structure contains the type of information that you want retrieved. The `size` field of this structure specifies the allocated size of the value buffer. Upon return, the `size` field specifies the actual size of the information that was retrieved, even if it was truncated to fit into the buffer.
- **ioTransIdP**  Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**  Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**  The following fields are updated in the `TelEventType` event that is sent when the operation completes:
WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

**Comments**

Call this function to retrieve information about the currently connected phone.

The retrieved information is stored into the value field of the `TelInfGetformationType` structure passed to this function in the `ioInfoP` parameter. If the value field buffer is too small to contain the complete information, the value is truncated and this function returns the `telErrBufferSize` error. The size field of the structure is always updated to contain the actual size of the retrieved information.

Before using this function, you should verify that it is available by calling the `TelIsInfServiceAvailable` macro.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.
**TelIsCfgServiceAvailable**

**Purpose**
A macro that determines if the configuration service set is available in the current environment.

**Declared In**
TelephonyMgr.h

**Prototype**
TelIsCfgServiceAvailable (iRefnum, iAppId, ioTransIdP)

**Parameters**
- iRefnum The telephony manager library reference number.
- iAppId The telephone application attachment identifier for your application.
- ioTransIdP Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns errNone if the service set is available, or an error code if not.

**Asynchronous Result**
The following fields are updated in the TelEventType event that is sent when the operation completes:

- returnCode errNone if the service set is available, or an error code if not.
- transId The transaction ID of the operation.
- paramP kTelCfgServiceId
- functionId kTelUrqIsServiceAvailableMessage

**Comments**
You need to call this macro before calling any function in the configuration service set, which is the family of functions that begin with the TelCfg prefix.
Telephony Basic Services
Telephony Functions

The configuration service set functions are documented in Chapter 72, “Telephony SMS.”

Compatibility
Implemented only if 4.0 New Feature Set is present.

See Also
TelIs<FunctionName>Supported

TellsDtcServiceAvailable

Purpose
A macro that determines if the data calls service set is available in the current environment.

Declared In
TelephonyMgr.h

Prototype
TelIsDtcServiceAvailable (iRefnum, iAppId, ioTransIdP)

Parameters
-> iRefnum The telephony manager library reference number.
-> iAppId The telephone application attachment identifier for your application.
<-> ioTransIdP Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result
Returns errNone if the service set is available, or an error code if not.

Asynchronous Result
The following fields are updated in the TelEvent_Type event that is sent when the operation completes:

returnCode errNone if the service set is available, or an error code if not.
transId The transaction ID of the operation.
Comments
You need to call this macro before calling any function in the data calls service set, which is the family of functions that begin with the TelDtc prefix.

The data calls service set functions are documented in Chapter 71, “Telephony Calls.”

Compatibility
Implemented only if 4.0 New Feature Set is present.

See Also
TelIs<FunctionName>Supported

TellIsEmcServiceAvailable

Purpose
A macro that determines if the emergency calls service set is available in the current environment.

Declared In
TelephonyMgr.h

Prototype
TelIsEmcServiceAvailable (iRefnum, iAppId, ioTransIdP)

Parameters
-> iRefnum The telephony manager library reference number.

-> iAppId The telephone application attachment identifier for your application.

<=-> ioTransIdP Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result
Returns errNone if the service set is available, or an error code if not.
Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode` errNone if the service set is available, or an error code if not.
- `transId` The transaction ID of the operation.
- `paramP` `kTelEmcServiceId`
- `functionId` `kTelUrqIsServiceAvailableMessage`

Comments

You need to call this macro before calling any function in the emergency calls service set, which is the family of functions that begin with the `TelEmc` prefix.

The emergency calls service set functions are documented in Chapter 71, “Telephony Calls.”

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelIs<FunctionName>Supported`

TellsInfServiceAvailable

Purpose

A macro that determines if the information service set is available in the current environment.

Declared In

`TelephonyMgr.h`

Prototype

`TelIsInfServiceAvailable (iRefnum, iAppId, ioTransIdP)`

Parameters

- `iRefnum` The telephony manager library reference number.
- `iAppId` The telephone application attachment identifier for your application.
- `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously.
If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

### Synchronous Result
Returns errNone if the service set is available, or an error code if not.

### Asynchronous Result
The following fields are updated in the **TelEventType** event that is sent when the operation completes:

- **returnCode**: errNone if the service set is available, or an error code if not.
- **transId**: The transaction ID of the operation.
- **paramP**: kTelInfServiceId
- **functionId**: kTelUrqIsServiceAvailableMessage

### Comments
You need to call this macro before calling any function in the information service set, which is the family of functions that begin with the `TelInf` prefix.

The information service set functions are documented in this chapter.

### Compatibility
Implemented only if **4.0 New Feature Set** is present.

### See Also
[TelIs<FunctionName>Supported](#)
**TellIsNwkServiceAvailable**

**Purpose**
A macro that determines if the network service set is available in the current environment.

**Declared In**
TelephonyMgr.h

**Prototype**
TelIsNwkServiceAvailable (iRefnum, iAppId, ioTransIdP)

**Parameters**
- iRefnum
  The telephony manager library reference number.
- iAppId
  The telephone application attachment identifier for your application.
- ioTransIdP
  Set the value of this parameter to NULL to cause the function to execute synchronously.
  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns errNone if the service set is available, or an error code if not.

**Asynchronous Result**
The following fields are updated in the TelEventType event that is sent when the operation completes:

- **returnCode**
  errNone if the service set is available, or an error code if not.
- **transId**
  The transaction ID of the operation.
- **paramP**
  kTelNwkServiceId
- **functionId**
  kTelUrqIsServiceAvailableMessage

**Comments**
You need to call this macro before calling any function in the network service set, which is the family of functions that begin with the TelNwk prefix.
The network service set functions are documented in Chapter 70, “Telephony Network.”

Compatibility
Implemented only if 4.0 New Feature Set is present.

See Also
TelIs<FunctionName>Supported

TellIsOemServiceAvailable

Purpose
A macro that determines if the OEM service set is available in the current environment.

Declared In
TelephonyMgr.h

Prototype
TelIsOemServiceAvailable (iRefnum, iAppId, ioTransIdP)

Parameters
-> iRefnum The telephony manager library reference number.
-> iAppId The telephone application attachment identifier for your application.
<- ioTransIdP Set the value of this parameter to NULL to cause the function to execute synchronously.

Synchronous Result
Returns errNone if the service set is available, or an error code if not.

Asynchronous Result
The following fields are updated in the TelEventType event that is sent when the operation completes:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>returnCode</td>
<td>errNone if the service set is available, or an error code if not.</td>
</tr>
<tr>
<td>transId</td>
<td>The transaction ID of the operation.</td>
</tr>
</tbody>
</table>
paramP kTelOemServiceId
functionId kTelUrqIsServiceAvailableMessage

Comments You need to call this macro before calling any function in the OEM service set, which is the family of functions that begin with the TelOem prefix.
The OEM service set functions are documented in this chapter.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also TelIs<FunctionName>Supported

TellIsPhbServiceAvailable

Purpose A macro that determines if the phone book service set is available in the current environment.

Declared In TelephonyMgr.h

Prototype TelIsPhbServiceAvailable (iRefnum, iAppId, ioTransIdP)

Parameters
-> iRefnum The telephony manager library reference number.
-> iAppId The telephone application attachment identifier for your application.
<-> ioTransIdP Set the value of this parameter to NULL to cause the function to execute synchronously.
If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns errNone if the service set is available, or an error code if not.
Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode` errNone if the service set is available, or an error code if not.
- `transId` The transaction ID of the operation.
- `paramP` `kTelPhbServiceId`
- `functionId` `kTelUrqIsServiceAvailableMessage`

Comments

You need to call this macro before calling any function in the phone book service set, which is the family of functions that begin with the `TelPhb` prefix.

The phone book service set functions are documented in Chapter 73, “Telephony Phone Book.”

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelIs<FunctionName>Supported`

**TellIsPhoneConnected**

Purpose

Determines if a phone is connected.

Declared In

`TelephonyMgr.h`

Prototype

`Err TelIsPhoneConnected(UInt16 iRefnum, TelAppID iAppId, UInt16 *ioTransIdP)`

Parameters

- `-> iRefnum` The telephony manager library reference number.
- `-> iAppId` The telephone application attachment identifier for your application.
- `<-> ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously.
If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**

Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**

The following fields are updated in the TelEventType event that is sent when the operation completes:

- **returnCode**: errNone upon success or an error code upon failure.
- **transId**: The transaction ID of the operation.
- **paramP**: A NULL pointer.
- **functionId**: kTelUrqIsPhoneConnectedMessage

**Comments**

Call this function to determine if there is currently a phone connected.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.

### TellsPowServiceAvailable

**Purpose**

A macro that determines if the power services set is available in the current environment.

**Declared In**

TelephonyMgr.h

**Prototype**

TelIsPowServiceAvailable (iRefnum, iAppId, ioTransIdP)

**Parameters**

- `-> iRefnum`: The telephony manager library reference number.
- `-> iAppId`: The telephone application attachment identifier for your application.
Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**

Returns errNone if the service set is available, or an error code if not.

**Asynchronous Result**

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode`: errNone if the service set is available, or an error code if not.
- `transId`: The transaction ID of the operation.
- `paramP`: kTelPowServiceId
- `functionId`: kTelUrqIsServiceAvailableMessage

**Comments**

You need to call this macro before calling any function in the power service set, which is the family of functions that begin with the `TelPow` prefix.

The power service set functions are documented in this chapter.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.

**See Also**

TelIs<FunctionName>Supported
TelIsSmsServiceAvailable

**Purpose**
A macro that determines if the Short Message Service (SMS) service set is available in the current environment.

**Declared In**
TelephonyMgr.h

**Prototype**
TelIsSmsServiceAvailable (iRefnum, iAppId, ioTransIdP)

**Parameters**
- `-> iRefnum` The telephony manager library reference number.
- `-> iAppId` The telephone application attachment identifier for your application.
- `<-> ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns *errNone* if the service set is available, or an error code if not.

**Asynchronous Result**
The following fields are updated in the *TelEventType* event that is sent when the operation completes:

- `returnCode` *errNone* if the service set is available, or an error code if not.
- `transId` The transaction ID of the operation.
- `paramP` `kTelSmsServiceId`
- `functionId` `kTelUrqIsServiceAvailableMessage`

**Comments**
You need to call this macro before calling any function in the SMS service set, which is the family of functions that begin with the *TelSms* prefix.
The SMS service set functions are documented in Chapter 72, “Telephony SMS.”

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
TelIs<FunctionName>Supported

**TelIsSndServiceAvailable**

**Purpose**
A macro that determines if the sound service set is available in the current environment.

**Declared In**
TelephonyMgr.h

**Prototype**
TelIsSndServiceAvailable (iRefnum, iAppId, ioTransIdP)

**Parameters**
- iRefnum
  The telephony manager library reference number.
- iAppId
  The telephone application attachment identifier for your application.
- ioTransIdP
  Set the value of this parameter to NULL to cause the function to execute synchronously.
  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns errNone if the service set is available, or an error code if not.

**Asynchronous Result**
The following fields are updated in the TelEventType event that is sent when the operation completes:

- returnCode
  errNone if the service set is available, or an error code if not.
- transId
  The transaction ID of the operation.
paramP kTelSndServiceId
functionId kTelUrqIsServiceAvailableMessage

Comments You need to call this macro before calling any function in the sound service set, which is the family of functions that begin with the TelSnd prefix. The sound service set functions are documented in this chapter.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also TelIs<FunctionName>Supported

**TellSpcServiceAvailable**

**Purpose** A macro that determines if the speech telephone call service set is available in the current environment.

**Declared In** TelephonyMgr.h

**Prototype** TellSpcServiceAvailable (iRefnum, iAppId, ioTransIdP)

**Parameters**

- `iRefnum` The telephony manager library reference number.
- `iAppId` The telephone application attachment identifier for your application.
- `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the service set is available, or an error code if not.
The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode` errNone if the service set is available, or an error code if not.
- `transId` The transaction ID of the operation.
- `paramP` `kTelSpcServiceId`
- `functionId` `kTelUrqIsServiceAvailableMessage`

You need to call this macro before calling any function in the speech telephone call service set, which is the family of functions that begin with the `TelSpc` prefix.

The speech telephone call service set functions are documented in Chapter 71, "Telephony Calls."

Implemented only if 4.0 New Feature Set is present.

A macro that determines if the security service set is available in the current environment.

Declared In `TelephonyMgr.h`

Prototype `TelIsStyServiceAvailable (iRefnum, iAppId, ioTransIdP)`

- `iRefnum` The telephony manager library reference number.
- `iAppId` The telephone application attachment identifier for your application.
- `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously.
If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

### Synchronous Result
Returns `errNone` if the service set is available, or an error code if not.

### Asynchronous Result
The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode`  : `errNone` if the service set is available, or an error code if not.
- `transId`  : The transaction ID of the operation.
- `paramP`  : `kTelStyServiceId`
- `functionId`  : `kTelUrqIsServiceAvailableMessage`

### Comments
You need to call this macro before calling any function in the security service set, which is the family of functions that begin with the `TelSty` prefix.

The security service set functions are documented in Chapter 70, “Telephony Network.”

### Compatibility
Implemented only if 4.0 New Feature Set is present.

### See Also
`TelIs<FunctionName>Supported`
TelMatchPhoneDriver

**Purpose**
Determines if the currently selected driver matches the connected phone.

**Declared In**
TelephonyMgr.h

**Prototype**
```
Err TelMatchPhoneDriver(UInt16 iRefnum, TelAppID iAppId, UInt16 *ioTransIdP)
```

**Parameters**
- `iRefnum` The telephony manager library reference number.
- `iAppId` The telephone application attachment identifier for your application.
- `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the **TelEventType** event that is sent when the operation completes:

- `returnCode` errNone upon success or an error code upon failure
- `transId` The transaction ID of the operation.
- `paramP` A NULL pointer.
- `functionId` kTelUrqMatchPhoneDriverMessage

**Compatibility**
Implemented only if 4.0 New Feature Set is present.
**TelOemCall**

**Purpose**
Pass a call to an OEM function.

**Declared In**
TelephonyMgr.h

**Prototype**
```
Err TelOemCall(UInt16 iRefnum, TelAppID iAppId, TelOemCallType *ioParamP, UInt16 *ioTransIdP)
```

**Parameters**
- `iRefnum` The telephony manager library reference number.
- `iAppId` The telephone application attachment identifier for your application.
- `ioParamP` A pointer to a `TelOemCallType` structure that contains information about the OEM function call.
- `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode` `errNone` upon success or an error code upon failure.
- `transID` matches the output value of the `ioTransIdP` parameter.
functionId  matches the function ID in the `TelOemCallType` structure passed to this function in the `ioParamP` parameter

paramId  points to the `TelOemCallType` structure passed to this function in the `ioParamP` parameter

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

**Comments**

Call this function to send a request to an OEM function. The calling function and the OEM function are responsible for coordinating the parameter block that is passed in the `TelOemCallType` structure.

Before using this function, you should verify that it is available by calling the `TelIsOemServiceAvailable` macro.

**Compatibility**

Implemented only if [4.0 New Feature Set](#) is present.

**TelOpen**

**Purpose**

Open the telephony manager API to initialize telephony services and launch the telephony task.

**Declared In**

`TelephonyMgr.h`

**Prototype**

```c
Err TelOpen(UInt16 iRefnum, UInt32 iVersnum, TelAppID *oAppIdP)
```

**Parameters**

- `iRefnum`  The telephony manager library reference number.
- `iVersnum`  The version number of the shared library for which your application was developed.
<oAppIdP> A pointer to an application ID value. Upon return, this is the application ID that you supply as a parameter to the any other telephony functions that you call.

**Result** Returns `errNone` if the function was successful or returns an error code if not successful. The following errors can occur:

- the telephony task could not be found (`telErrTTaskNotFound`)
- the telephony task could not be launched (`telErrTTaskNotRunning`)
- the phone driver could not be found
- the shared library version is not valid

**Comments** You can only call this function synchronously. You must call this function before calling any other telephony manager functions.

You can specify the current version of the shared library by using the `kTelMgrVersion` constant as the value of the `iVersnum` parameter.

**Compatibility** Implemented only if 4.0 New Feature Set is present.

**See Also** TelClose

---

**TelOpenPhoneConnection**

**Purpose** Open communications with the connected phone.

**Declared In** TelephonyMgr.h

**Prototype**

```
Err TelOpenPhoneConnection(UInt16 iRefnum, TelAppID iAppId, UInt16 *ioTransIdP)
```

**Parameters**

- `iRefnum` The telephony manager library reference number.
- `iAppId` The telephone application attachment identifier for your application.
Telephony Basic Services
Telephony Functions

Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns errNone if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the TelEventType event that is sent when the operation completes:

- returnCode errNone upon success or an error code upon failure.
- transId The transaction ID of the operation.
- paramP A NULL pointer.
- functionId kTelUrqOpenPhoneConnectionMessage

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also TelClosePhoneConnection

TelPowGetBatteryStatus

Purpose Retrieves the status of the phone’s battery.

Declared In TelephonyMgr.h

Prototype Err TelPowGetBatteryStatus(UInt16 iRefnum, TelAppID iAppId, UInt8 *oStatusP, UInt16 *ioTransIdP)

Parameters
- -> iRefnum The telephony manager library reference number.
- -> iAppId The telephone application attachment identifier for your application.
<- oStatusP    A pointer to an unsigned byte value. Upon return, this is the battery status value, which is one of the Battery Status Constants.

<-> ioTransIdP  Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the TelEventType event that is sent when the operation completes:

- **returnCode**  errNone upon success or an error code upon failure.
- **transId**  The transaction ID of the operation.
- **paramP**  Points to the unsigned integer value passed to this function in the oStatusP parameter.
- **functionId**  kTelPowBatteryStatusMessage

**WARNING!**  When using this function asynchronously, you must ensure that the value referenced by oStatusP remains in memory until the asynchronous call completes.

**Comments**
Before using this function, you should verify that it is available by calling the TelIsPowServiceAvailable macro.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**  TelPowGetPowerLevel
TelPowGetPowerLevel

Purpose
Retrieve the current level of the phone battery, as a percentage value.

Declared In
TelephonyMgr.h

Prototype
Err TelPowGetPowerLevel(UInt16 iRefnum, TelAppID iAppId, UInt8 *oPowerP, UInt16 *ioTransIdP)

Parameters
-> iRefnum The telephony manager library reference number.
-> iAppId The telephone application attachment identifier for your application.
<- oPowerP A pointer to an unsigned byte value. Upon return, this is the battery percentage value.
<--> ioTransIdP Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result
Returns errNone if the function was successful or returns an error code if not successful.

Asynchronous Result
The following fields are updated in the TelEventType event that is sent when the operation completes:

returnCode errNone upon success or an error code upon failure.
transId The transaction ID of the operation.
pParamP Points to the unsigned integer value passed to this function in the oPowerP parameter.
functionId kTelPowGetPowerLevelMessage
**WARNING!** When using this function asynchronously, you must ensure that the value referenced by `oPowerP` remains in memory until the asynchronous call completes.

**Comments**

The returned percentage value is an integer value between 0 and 100.

Before using this function, you should verify that it is available by calling the `TelIsPowServiceAvailable` macro.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.

**See Also**

`TelPowGetBatteryStatus`

---

**TelPowSetPhonePower**

**Purpose**

Turns the phone on or off.

**Declared In**

`TelephonyMgr.h`

**Prototype**

```c
Err TelPowSetPhonePower(UInt16 iRefnum, TelAppID iAppId, Boolean iPowerOn)
```

**Parameters**

- `-> iRefnum` The telephony manager library reference number.
- `-> iAppId` The telephone application attachment identifier for your application.
- `-> iPowerOn` Set to `true` to turn the phone on, and set to `false` to turn the phone off.

**Result**

Returns `errNone` if the function was successful and an error code if not.

**Comments**

This function can only be called synchronously.

Before using this function, you should verify that it is available by calling the `TelIsPowServiceAvailable` macro.
This function corresponds to the kTelPowSetPhonePowerMessage function ID value.

**Compatibility**  Implemented only if 4.0 New Feature Set is present.

**TelSendCommandString**

**Purpose** Sends a command string to the phone or to the network.

**Declared In** TelephonyMgr.h

**Prototype**

```c
Err TelSendCommandString(UInt16 iRefnum,
TelAppID iAppId,
TelSendCommandStringType *ioParamP,
UInt16 *ioTransIdP)
```

**Parameters**

- **-> iRefnum** The telephony manager library reference number.
- **-> iAppId** The telephone application attachment identifier for your application.
- **<-> ioParamP** A pointer to a command string structure of type TelSendCommandStringType.
- **<-> ioTransIdP** Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**

Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**

The following fields are updated in the TelEventType event that is sent when the operation completes:
Telephony Basic Services
Telephony Functions

returnCode
errNone upon success or an error code upon failure.

transId
The transaction ID of the operation.

paramP
Points to the TelSendCommandStringType structure passed to this function in the ioParam parameter.

functionId
Matches the function ID in the TelOemCallType referenced by the ioParamP structure.

WARNING! When using this function asynchronously, you must ensure that the structure referenced by ioParamP remains in memory until the asynchronous call completes.

Compatibility
Implemented only if 4.0 New Feature Set is present.

TelSndMute

Purpose
Mute or un-mute an active telephone call.

Declared In
TelephonyMgr.h

Prototype
Err TelSndMute(UInt16 iRefnum, TelAppID iAppId, Boolean iMuteOn, UInt16 *ioTransIdP)

Parameters
-> iRefnum
The telephony manager library reference number.

-> iAppId
The telephone application attachment identifier for your application.

-> iMuteOn
Set to true to mute the telephone call, or set to false to unmute the telephone call.

<-> ioTransIdP
Set the value of this parameter to NULL to cause the function to execute synchronously.
If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode`  
  `errNone` upon success or an error code upon failure.
- `transId`  
  The transaction ID of the operation.
- `paramP`  
  Points to the Boolean value passed to this function in the `iMuteOn` parameter.
- `functionId`  
  `kTelSndMuteMessage`

**Comments**
Before using this function, you should verify that it is available by calling the `TelIsSndServiceAvailable` macro.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
`TelSndPlayKeyTone`

---

**TelSndPlayKeyTone**

**Purpose**
Play a keytone sound on the phone.

**Declared In**
`TelephonyMgr.h`

**Prototype**
```c
Err TelSndPlayKeyTone(UInt16 iRefnum, 
TelAppID iAppId, TelSndPlayKeyToneType *iParamP, 
UInt16 *ioTransIdP)
```

**Parameters**
- `-> iRefnum`  
  The telephony manager library reference number.
The telephone application attachment identifier for your application.

A pointer to a `TelSndPlayKeyToneType` structure that describes the tone to play.

Set the value of this parameter to `NULL` to cause the function to execute synchronously. If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**

Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode`  `errNone` upon success or an error code upon failure.
- `transId`      The transaction ID of the operation.
- `paramP`      Points to the `TelSndPlayKeyToneType` structure passed to this function in the `iKeyToneP` parameter.
- `functionId`  `kTelSndPlayKeyTone`  

**WARNING!**  When using this function asynchronously, you must ensure that the structure referenced by `iParamP` remains in memory until the asynchronous call completes.

**Comments**

Before using this function, you should verify that it is available by calling the `TelIsSndServiceAvailable` macro.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.

**See Also**  `TelSndStopKeyTone`
### TelSndStopKeyTone

**Purpose**
Stop the playing of a keytone sound on the phone.

**Declared In**
TelephonyMgr.h

**Prototype**
```c
Err TelSndStopKeyTone(UInt16 iRefnum, TelAppID iAppId, UInt16 *ioTransIdP)
```

**Parameters**
- **-> iRefnum**
The telephony manager library reference number.
- **-> iAppId**
The telephone application attachment identifier for your application.
- **<-> ioTransIdP**
Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns **errNone** if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the **TelEventType** event that is sent when the operation completes:

- **returnCode**
  errNone upon success or an error code upon failure.
- **transId**
The transaction ID of the operation.
- **paramP**
  A NULL pointer.
- **functionId**
kTelSndStopKeyToneMessage

**Comments**
Before using this function, you should verify that it is available by calling the **TelIsSndServiceAvailable** macro.
Compatibility
Implemented only if 4.0 New Feature Set is present.

See Also
TelSndPlayKeyTone

Feature Support Functions
This section describes the functions that you can call to determine if a specific feature or function is supported in the current operating environment.

Tells<FunctionName>Supported

Purpose
Determines if the specified function is supported.

Declared In
TelephonyMgr.h

Prototype
TelIs<FunctionName>Supported (iRefnum, iAppId, ioTransIdP)

Parameters
- iRefnum
  The telephony manager library reference number.
- iAppId
  The telephone application attachment identifier for your application.
- ioTransIdP
  Set the value of this parameter to NULL to cause the function to execute synchronously.
  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result
Returns errNone if the specified function is supported.

Asynchronous Result
The following fields are updated in the TelEventType event that is sent when the operation completes:
This is a family of synchronous macros that test if a specific function is available in the current environment. To use the macro, substitute a function name for the <FunctionName> portion of the macro name. You can substitute any Telephony Manager function name; for a complete list of the Telephony Manager functions, see “Summary of Telephony Manager” on page 235 in Palm OS Programmer’s Companion, vol. II, Communications.

For example, to determine if the TelNwkGetSignalLevel function is available in the current environment, call the TelIsNwkGetSignalLevelSupported macro.

**NOTE:** A service set can be available without all of its functions being available. Thus, if the TelIs<ServiceSet>Available macro returns true for a specific service set, you know that the service set is available, but you need to call TelIs<FunctionName>Supported to determine if a specific function is available.

This macro corresponds to the kTelUrqIsFunctionSupportedMessage function ID value.

| returnCode | errNone if the specified function is supported, or an error code if not. |
| transId    | The transaction ID of the operation.                                    |
| paramP     | The function ID of the function for which you are testing. For example, if you call TelIsCgfGetSmsCenterSupported, the value of this field is kTelCgfGetSmsCenterMessage. The function ID value for each function is described in the documentation for the function. |
| functionId | kTelUrqIsFunctionSupportedMessage                                      |
**Telephony Basic Services**

**Feature Support Functions**

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
TelIs<ServiceSet>Available

**Tells<ServiceSet>Available**

**Purpose**
Determines if the specified service set is available.

**Declared In**
TelephonyMgr.h

**Prototype**
TelIs<ServiceSet>Available (iRefnum, iAppId, ioTransIdP)

**Parameters**
- `iRefnum`  
The telephony manager library reference number.
- `iAppId`  
The telephone application attachment identifier for your application.
- `ioTransIdP`  
Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns errNone if the service set is available.

**Asynchronous Result**
The following fields are updated in the TelEventType event that is sent when the operation completes:

- `returnCode`  
errNone if the service set is available, or an error code if not.

- `transId`  
The transaction ID of the operation.
paramP The service ID of the service set for which you are testing. For example, if you call TelIsCfgServiceAvailable, the value of this field is kTelCfgServiceId.

The service IDs are described in Service Set Constants.

functionId kTelUrqIsServiceAvailableMessage

Comments This is a family of synchronous macros that test if a specific service set is available. You must call the appropriate set availability function before calling a function in the set.

NOTE: A service set can be available without all of its functions being available. You can use this macro to determine the availability of a specific service set, which you might use to determine the configuration of your applications' user interface.

To test if a specific function is supported, use the TelIs<FunctionName>Supported macro.

You can call these specific macros to determine if the service set is available:

- TelIsCfgServiceAvailable to determine if the configuration service set is available.
- TelIsDtcServiceAvailable to determine if the data calls service set is available.
- TelIsEmcServiceAvailable to determine if the emergency calls service set is available.
- TelIsInfServiceAvailable to determine if the information service set is available.
- TelIsNwkServiceAvailable to determine if the network service set is available.
- TelIsOemServiceAvailable to determine if the OEM service set is available.
- TelIsPhbServiceAvailable to determine if the phone book service set is available.
Telephony Basic Services
Feature Support Functions

- `TelIsPowServiceAvailable` to determine if the power service set is available.
- `TelIsSmsServiceAvailable` to determine if the SMS service set is available.
- `TelIsSndServiceAvailable` to determine if the sound service set is available.
- `TelIsSpcServiceAvailable` to determine if the speech calls service set is available.
- `TelIsStyServiceAvailable` to determine if the security service set is available.

Each of these macros corresponds to the `kTelUrqIsServiceSupportedMessage` function ID value.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelIs<FunctionName>Supported`
Telephony Security and Configuration

This chapter describes the telephony security and configuration service sets of the telephony API.

For more information about the telephony manager basic services and the different service sets, see Chapter 68, “Telephony Basic Services.”

This chapter describes:
• Telephony Security and Configuration Data Structures
• Telephony Security and Configuration Constants
• Telephony Security and Configuration Functions

For more information about using the telephony manager, see Chapter 10, “Telephony Manager,” in Palm OS Programmer’s Companion, vol. II, Communications.

Telephony Security and Configuration Data Structures

This section describes the data structures used with the telephony security and configuration service sets portion of the telephony API.

TelCfgGetPhoneNumberType

The TelCfgGetPhoneNumber function uses a TelCfgGetPhoneNumberType structure to retrieve the connected phone dial number.
typedef struct _TelCfgGetPhoneNumberType
    UInt8    size;
    Char*    value;
} TelCfgGetPhoneNumberType

Field Descriptions

<-> size

The size of the value buffer.

When the structure is used as an input parameter, this is the allocated size, in bytes, of the value buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the value buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a telErrBufferSize error.

<- value

A buffer into which the dial number is stored.

Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a telErrBufferSize error.

TelCfgGetSmsCenterType

The TelCfgGetSmsCenter function uses a TelCfgGetSmsCenterType structure to retrieve the SMS service center dial number.

typedef struct _TelCfgGetSmsCenterType
    UInt8    size;
    Char*    value;
} TelCfgGetSmsCenterType
Field Descriptions

<-> size

The size of the value buffer.

When the structure is used as an input parameter, this is the allocated size, in bytes, of the value buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the value buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a telErrBufferSize error.

<-> value

A buffer into which the dial number is stored.

Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a telErrBufferSize error.

TelStyChangeAuthenticationType

You use the TelStyChangeAuthenticationType to change an authentication code with the TelStyChangeAuthenticationCode function.

```c
typedef struct _TelStyChangeAuthenticationType
{
  UInt    codeId;
  Char*   oldCode;
  Char*   newCode;
} TelStyChangeAuthenticationType
```

Field Descriptions

- > codeId

The ID of the authentication code to change.

- > oldCode

The previous value of the code.

- > newCode

The new value of the code.
Telephony Security and Configuration Constants

This section describes the constants used with the telephony security and configuration service sets of the telephony API.

Authentication State Constants

The authentication state constants describe the current authentication state of the mobile unit connection.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTelStyReady</td>
<td>0</td>
<td>No additional security information is expected.</td>
</tr>
<tr>
<td>kTelStyPin1CodeId</td>
<td>1</td>
<td>The PIN1 code is expected.</td>
</tr>
<tr>
<td>kTelStyPin2CodeId</td>
<td>2</td>
<td>The PIN2 code is expected.</td>
</tr>
<tr>
<td>kTelStyPuk1CodeId</td>
<td>3</td>
<td>The PUK1 code is expected.</td>
</tr>
<tr>
<td>kTelStyPuk2CodeId</td>
<td>4</td>
<td>The PUK2 code is expected.</td>
</tr>
<tr>
<td>kTelStyPhoneToSimCodeId</td>
<td>5</td>
<td>The phone-to-SIM code is expected.</td>
</tr>
<tr>
<td>kTelStyFirstOemCodeId</td>
<td>6</td>
<td>An OEM code is expected.</td>
</tr>
</tbody>
</table>

The constant kTelStyFirstOemCodeId specifies the first OEM authentication code. You can specify additional OEM codes by incrementing this value. For example, to specify the third OEM authentication code, use the following:

kTelStyFirstOemCodeId + 2

Telephony Security and Configuration Functions

This section describes the data structures used with the telephony security and configuration service sets portion of the telephony API.
**TelCfgGetPhoneNumber**

**Purpose**  Retrieve the connected telephone number.

**Declared In**  TelephonyMgr.h

**Prototype**  
```
Err TelCfgGetPhoneNumber(UInt16 iRefnum, 
TelAppID iAppId, 
TelCfgGetPhoneNumberType* ioParamP, 
UInt16* ioTransIdP)
```

**Parameters**

- `iRefnum`  The telephony manager library reference number.

- `iAppId`  The telephone application attachment identifier for your application.

- `ioParamP`  A pointer to a `TelCfgGetPhoneNumberType` structure that is filled in with the dial telephone number.

- `ioTransIdP`  Set the value of this parameter to `NULL` to cause the function to execute synchronously.
  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**  Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**  The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode`  `errNone` upon success or an error code upon failure.

- `transId`  The transaction ID of the operation.
paramP  Points to the `TelCfgGetPhoneNumberType` structure passed to this function in the `ioParamP` parameter.

`functionId` kTelCfgGetPhoneNumberMessage

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

**Comments** The connected dial telephone number is stored into the value field of the `TelCfgGetPhoneNumberType` structure referenced by `ioParamP`. If the value field buffer is too small to contain the complete telephone number, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The size field of the structure is always updated to contain the actual size of the complete telephone number.

Before using this function, you should verify that it is available by calling the `TelIsCfgServiceAvailable` macro.

**Compatibility** Implemented only if 4.0 New Feature Set is present.

**See Also** `TelCfgSetSmsCenter`, `TelSmsSendMessage`
TelCfgGetSmsCenter

**Purpose**  
Retrieve the SMS service center dial telephone number.

**Declared In**  
TelephonyMgr.h

**Prototype**  
Err TelCfgGetSmsCenter(UInt16 iRefnum,  
TelAppID iAppId,  
TelCfgGetSmsCenterType* ioParamP,  
UInt16* ioTransIdP)

**Parameters**

- **-> iRefnum**  
The telephony manager library reference number.

- **-> iAppId**  
The telephone application attachment identifier for your application.

- **<-> ioParamP**  
A pointer to a *TelCfgGetSmsCenterType* structure that is filled in with the dial telephone number.

- **<-> ioTransIdP**  
Set the value of this parameter to NULL to cause the function to execute synchronously.  
If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**  
Returns *errNone* if the function was successful or returns an error code if not successful.

**Asynchronous Result**  
The following fields are updated in the *TelEventType* event that is sent when the operation completes:

- **returnCode**  
*errNone* upon success or an error code upon failure.

- **transId**  
The transaction ID of the operation.
paramP Points to the `TelCfgGetSmsCenterType` structure passed to this function in the `ioParamP` parameter.

functionId kTelCfgGetSmsCenterMessage

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

**Comments**
The service center dial telephone number is stored into the value field of the `TelCfgGetSmsCenterType` structure referenced by `ioParamP`. If the value field buffer is too small to contain the complete telephone number, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The size field of the structure is always updated to contain the actual size of the complete telephone number.

Before using this function, you should verify that it is available by calling the `TelIsCfgServiceAvailable` macro.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
`TelCfgSetSmsCenter`, `TelSmsSendMessage`

### `TelCfgSetSmsCenter`

**Purpose**
Set the SMS service center dial telephone number.

**Declared In**
`TelephonyMgr.h`

**Prototype**
```c
Err TelCfgSetSmsCenter(UInt16 iRefnum, TelAppID iAppId, const Char* iDialNumberP, UInt16* ioTransIdP)
```

**Parameters**
- `iRefnum` The telephony manager library reference number.
Telephony Security and Configuration

Telephony Security and Configuration Functions

- `iAppId` The telephone application attachment identifier for your application.

- `iDialNumberP` A pointer to the null-terminated dial telephone number string for the SMS service center.

- `ioTransIdP` Set the value of this parameter to `NULL` to cause the function to execute synchronously. If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**

Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode` `errNone` upon success or an error code upon failure.

- `transId` The transaction ID of the operation.

- `paramP` Points to the string passed to this function in the `iDialNumberP` parameter.

- `functionId` `kTelCfgSetSmsCenterMessage`

---

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by `iDialNumberP` remains in memory until the asynchronous call completes.

**Comments**

Before using this function, you should verify that it is available by calling the `TelIsCfgServiceAvailable` macro.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.

**See Also**

`TelCfgGetSmsCenter`, `TelSmsSendMessage`
**TelStyChangeAuthenticationCode**

**Purpose**
Change the value of an authentication code. Note that you can only use this function with GSM networks.

**Declared In**
TelephonyMgr.h

**Prototype**
```
Err TelStyChangeAuthenticationCode(UInt16 iRefnum,
TelAppID iAppId,
TelStyChangeAuthenticationType* iParamP,
UInt16* ioTransIdP)
```

**Parameters**
- **-> iRefnum** The telephony manager library reference number.
- **-> iAppId** The telephone application attachment identifier for your application.
- **-> iParamP** A pointer to a `TelStyChangeAuthenticationType` structure that contains the old and new authentication code values.
- **<- ioTransIdP** Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- **returnCode** `errNone` upon success or an error code upon failure.
- **transId** The transaction ID of the operation.
paramP     Points to the TelStyChangeAuthenticationType structure passed to this function in the iParamP parameter.

functionId kTelStyChangeAuthenticationCodeMessage

WARNING! When using this function asynchronously, you must ensure that the structure referenced by iParamP remains in memory until the asynchronous call completes.

Comments Before using this function, you should verify that it is available by calling the TelIsStyServiceAvailable macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also TelStyEnterAuthenticationCode

TelStyEnterAuthenticationCode

Purpose Enter the authentication code for which the phone is currently waiting. Note that you can only use this function with GSM networks.

Declared In TelephonyMgr.h

Prototype Err TelStyEnterAuthenticationCode(UInt16 iRefnum, TelAppID iAppId, const Char* iCodeP, UInt16* ioTransIdP)

Parameters

- -> iRefnum The telephony manager library reference number.
- -> iAppId The telephone application attachment identifier for your application.
- -> iCodeP A pointer to the null-terminated authentication code string to send to the phone.
Telephony Security and Configuration
Telephony Security and Configuration Functions

ioTransIdP  Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result
Returns errNone if the function was successful or returns an error code if not successful.

Asynchronous Result
The following fields are updated in the TelEventType event that is sent when the operation completes:

- returnCode  errNone upon success or an error code upon failure.
- transId  The transaction ID of the operation.
- paramP  Points to the string passed to this function in the iCodeP parameter.
- functionId  kTelStyEnterAuthenticationCodeMessage

WARNING!  When using this function asynchronously, you must ensure that the string referenced by iCodeP remains in memory until the asynchronous call completes.

Comments
Before using this function, you should verify that it is available by calling the TelIsStyServiceAvailable macro.

Compatibility
Implemented only if 4.0 New Feature Set is present.

See Also
TelStyChangeAuthenticationCode, TelStyGetAuthenticationState
**TelStyGetAuthenticationState**

**Purpose**
Returns the current state of user authentication.

**Declared In**
TelephonyMgr.h

**Prototype**
```
Err TelStyGetAuthenticationState(UInt16 iRefnum, TelAppID iAppId, UInt8* oStateP, UInt16* ioTransIdP)
```

**Parameters**
- `iRefnum` The telephony manager library reference number.
- `iAppId` The telephone application attachment identifier for your application.
- `oStateP` A pointer to an unsigned byte value. Upon return, this is the authentication state, which is one of the Authentication State Constants.
- `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the TelEventType event that is sent when the operation completes:

- `returnCode` errNone upon success or an error code upon failure.
- `transId` The transaction ID of the operation.
- `paramP` Points to the unsigned integer value passed to this function in the oStateP parameter.
- `functionId` kTelStyGetAuthenticationStateMessage
WARNING! When using this function asynchronously, you must ensure that the value referenced by oStateP remains in memory until the asynchronous call completes.

Comments
Before using this function, you should verify that it is available by calling the TelIsStyServiceAvailable macro.

Compatibility
Implemented only if 4.0 New Feature Set is present.

See Also
TelStyEnterAuthenticationCode
Telephony Network

This chapter describes the telephony network service set of the telephony API.

For more information about the telephony manager basic services and the different service sets, see Chapter 68, "Telephony Basic Services."

This chapter describes:

- **Telephony Network Data Structures**
- **Telephony Network Constants**
- **Telephony Network Functions**

For more information about using the telephony manager, see Chapter 10, "Telephony Manager," in *Palm OS Programmer’s Companion*, vol. II, Communications.

**Telephony Network Data Structures**

This section describes the data structures used with the telephony network service set portion of the telephony API.

**TelNwkGetLocationType**

You use the TelNwkGetLocationType structure with the TelNwkGetLocation function to retrieve information about the location of the phone.

```c
typedef struct _TelNwkGetLocationType
{
    Char*   value;
    UInt16  size;
} TelNwkGetLocationType
```
Field Descriptions

<- value  A buffer into which the current location string is stored. The format of this string is network-dependent.

Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a `telErrBufferSize` error.

<-> size  The size of the value string. When the structure is used as an input parameter, this is the allocated size, in bytes, of the value buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the value buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.

**TelNwkGetNetworkNameType**

You use the `TelNwkGetNetworkNameType` structure with the `TelNwkGetNetworkName` function to retrieve the name of the specified, registered network.

```c
typedef struct _TelNwkGetNetworkNameType
{
    UInt32   id;
    Char*    value;
    UInt16   size;
} TelNwkGetNetworkNameType
```
Field Descriptions

-> id

The network ID.

<-> value

A null-terminated string buffer into which the network name is stored.

Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a `telErrBufferSize` error.

<-> size

The size of the value string. When the structure is used as an input parameter, this is the allocated length, in bytes, of the value buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the value buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.

TelNwkGetNetworksType

You use the `TelNwkGetNetworksType` structure with the `TelNwkGetNetworks` function to retrieve the number of registered networks.

```c
typedef struct _TelNwkGetNetworksType
{
    UInt32*   networkIdP;
    UInt8     size;
} TelNwkGetNetworksType
```
Telephony Network Constants

This section describes the constants used with the telephony network service set of the telephony API, which include the following constant types:

- Network Type Constants
- Network Search Mode Constants

Network Type Constants

The network type constants describe the type of network connection.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTelNwkCDMA</td>
<td>0</td>
<td>A CDMA network.</td>
</tr>
<tr>
<td>kTelNwkGSM</td>
<td>1</td>
<td>A GSM network.</td>
</tr>
</tbody>
</table>
Network Search Mode Constants
The network search mode constants describe the search mode used to find a network.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTelNwkManualSearch</td>
<td>0</td>
<td>Manual network searching.</td>
</tr>
<tr>
<td>kTelNwkAutomaticSearch</td>
<td>1</td>
<td>Automatic network searching.</td>
</tr>
</tbody>
</table>

Telephony Network Functions
This section describes the data structures used with the telephony network service set portion of the telephony API.

TelNwkGetLocation

**Purpose**
Retrieve information about the location of the mobile unit.

**Declared In**
TelephonyMgr.h

**Prototype**
Err TelNwkGetLocation(UInt16 iRefnum, TelAppID iAppId, TelNwkGetLocationType* ioParamP, UInt16* ioTransIdP)

**Parameters**
- -> iRefnum The telephony manager library reference number.
- -> iAppId The telephone application attachment identifier for your application.
- <-> ioParamP A pointer to a TelNwkGetLocationType structure.
On input, the `size` field of this structure specifies the allocated size of the `value` buffer. Upon return, the `size` field specifies the actual size of the location string, even if it was truncated to fit into the buffer.

```c
<> ioTransIdP
```

Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

### Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

### Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode`  
  `errNone` upon success or an error code upon failure.

- `transId`  
  The transaction ID of the operation.

- `paramP`  
  Points to the `TelNwkGetLocationType` passed to this function in the `ioParamP` parameter.

- `functionId`  
  `kTelNwkGetLocationMessage`

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

### Comments

The location information string is stored into the `value` field of the `TelNwkGetLocationType` structure referenced by `ioParamP`. If the value buffer is too small to contain the complete string, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The `size` field of the structure is always updated to contain the actual size of the complete string.
Before using this function, you should verify that it is available by calling the `TelIsNwkServiceAvailable` macro.

**Compatibility**  Implemented only if 4.0 New Feature Set is present.

### TelNwkGetNetworkName

**Purpose**  Returns the name of a registered network.

**Declared In**  TelephonyMgr.h

**Prototype**  
```c
Err TelNwkGetNetworkName(UInt16 iRefnum,  
TelAppID iAppId,  
TelNwkGetNetworkNameType* ioParamP,  
UInt16* ioTransIdP)
```

**Parameters**

- `<-> iRefnum`  The telephony manager library reference number.
- `<-> iAppId`  The telephone application attachment identifier for your application.
- `<-> ioParamP`  A pointer to a `TelNwkGetNetworkNameType` structure that stores the network name. On input, the size field of this structure specifies the allocated size of the value buffer. Upon return, the size field specifies the actual size of the name string, even if it was truncated to fit into the buffer.
- `<-> ioTransIdP`  Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**  Returns `errNone` if the function was successful or returns an error code if not successful.
Asynchronous Result

The following fields are updated in the **TelEventType** event that is sent when the operation completes:

- `returnCode` errNone upon success or an error code upon failure.
- `transId` The transaction ID of the operation.
- `paramP` Points to the **TelNwkGetNetworkNameType** passed to this function in the `ioParamP` parameter.
- `functionId` kTelNwkGetNetworkNameMessage

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

Comments

The network name string is stored into the value field of the structure. If the value field buffer in the **TelNwkGetNetworkNameType** structure is too small to contain the complete string, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The size field is always updated to contain the actual size of the complete string.

The string that is returned in the value field of the structure referenced by `ioParamP` is network dependent.

On a GSM network, the result string is compliant with the AT 07.07 European Telecommunications Standards Institute (ETSI) standard for COPS and CREG commands. The result string contains the following elements:

- The network type, as returned by the **TelNwkGetNetworkType** function, and followed by a semicolon (',') character.
- The network operator, using the following syntax:
<area code> ';'; <network registration>
The <area code> value is the value retrieved by issuing the AT+CREG? command.
The <network registration> value is the value retrieved by issuing the AT+CREG? command.
Before using this function, you should verify that it is available by calling the TelIsNwkServiceAvailable macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also TelNwkGetNetworks, TelNwkGetSelectedNetwork

TelNwkGetNetworks

Purpose Retrieves information about the registered networks.

Declared In TelephonyMgr.h

Prototype Err TelNwkGetNetworks(UInt16 iRefnum, TelAppID iAppId, TelNwkGetNetworksType* ioParamP, UInt16* ioTransIdP)

Parameters -> iRefnum The telephony manager library reference number.

-> iAppId The telephone application attachment identifier for your application.

<-> ioParamP A pointer to a TelNwkGetNetworksType structure that stores the network IDs. On input, the size field of this structure contains the size, in elements, of the networkIdP array field.

Upon return, the networkIdP array contains the IDs of the registered networks, and the size field contains the number of IDs in the array.
Telephony Network
Telephony Network Functions

`ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

### Synchronous Result
Returns `errNone` if the function was successful or returns an error code if not successful.

### Asynchronous Result
The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode` errNone upon success or an error code upon failure.
- `transId` The transaction ID of the operation.
- `paramP` Points to the `TelNwkGetNetworksType` passed to this function in the `ioParamP` parameter.
- `functionId` `kTelNwkGetNetworkCountMessage`

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

### Comments
Before using this function, you should verify that it is available by calling the `TelIsNwkServiceAvailable` macro.

### Compatibility
Implemented only if 4.0 New Feature Set is present.

### See Also
- `TelNwkGetNetworkName`, `TelNwkGetNetworks`
TelNwkGetNetworkType

**Purpose**
Retrieve the type of the selected network.

**Declared In**
TelephonyMgr.h

**Prototype**
Err TelNwkGetNetworkType(UInt16 iRefnum, TelAppID iAppId, UInt8* oTypeP, UInt16* ioTransIdP)

**Parameters**
- **-> iRefnum**
The telephony manager library reference number.
- **-> iAppId**
The telephone application attachment identifier for your application.
- **<- oTypeP**
A pointer to an unsigned byte value. Upon return, this is the network type. This is one of Network Type Constants.
- **<-> ioTransIdP**
Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the TelEventType event that is sent when the operation completes:
- **returnCode**
  errNone upon success or an error code upon failure.
- **transId**
The transaction ID of the operation.
- **paramP**
  Points to the unsigned integer value passed to this function in the oTypeP parameter.
- **functionId**
  kTelNwkGetNetworkTypeMessage
**WARNING!** When using this function asynchronously, you must ensure that the value you pass for the oTypeP parameter remains in memory until the asynchronous call completes.

**Comments**
Before using this function, you should verify that it is available by calling the `TelIsNwkServiceAvailable` macro.

**Compatibility**
Implemented only if [4.0 New Feature Set](#) is present.

**TelNwkGetSearchMode**

**Purpose**
Returns the current network search mode.

**Declared In**
TelephonyMgr.h

**Prototype**
```
Err TelNwkGetSearchMode(UInt16 iRefnum, TelAppID iAppId, UInt8* oModeP, UInt16* ioTransIdP)
```

**Parameters**
- **iRefnum**
  The telephony manager library reference number.
- **iAppId**
  The telephone application attachment identifier for your application.
- **oModeP**
  A pointer to an unsigned byte value. Upon return, this is the type of search mode that is currently being used. This is one of the [Network Search Mode Constants](#).
- **ioTransIdP**
  Set the value of this parameter to NULL to cause the function to execute synchronously.
  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
Synchronous Result

Returns errNone if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the TelEventType event that is sent when the operation completes:

- **returnCode**: errNone upon success or an error code upon failure.
- **transId**: The transaction ID of the operation.
- **paramP**: Points to the unsigned integer value passed to this function in the oModeP parameter.
- **functionId**: kTelNwkGetSearchModeMessage

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by oModeP remains in memory until the asynchronous call completes.

Comments

Before using this function, you should verify that it is available by calling the TelIsNwkServiceAvailable macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

TelNwkSetSearchMode

TelNwkGetSelectedNetwork

**Purpose**

Retrieve the network identifier of the currently selected network.

Declared In

TelephonyMgr.h

Prototype

Err TelNwkGetSelectedNetwork(UInt16 iRefnum, TelAppID iAppId, UInt32* oNetworkIdP, UInt16* ioTransIdP)

Parameters

- **-> iRefnum**: The telephony manager library reference number.
Telephony Network Functions

-> iAppId
   The telephone application attachment identifier for your application.

<- oNetworkIdP
   A pointer to an unsigned integer value. Upon return, this is the identifier of the currently selected network.

<-> ioTransIdP
   Set the value of this parameter to NULL to cause the function to execute synchronously.
   If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result
   Returns errNone if the function was successful or returns an error code if not successful.

Asynchronous Result
   The following fields are updated in the TelEventType event that is sent when the operation completes:

   returnCode  errNone upon success or an error code upon failure.
   transId      The transaction ID of the operation.
   paramP       Points to the unsigned integer value passed to this function in the oNetworkIdP parameter.
   functionId   kTelNwkGetSelectedNetworkMessage

WARNING!  When using this function asynchronously, you must ensure that the value referenced by oNetworkP remains in memory until the asynchronous call completes.

Comments
   Before using this function, you should verify that it is available by calling the TelIsNwkServiceAvailable macro.
Compatibility Implemented only if 4.0 New Feature Set is present.

See Also TelNwkGetNetworkName, TelNwkGetNetworks, TelNwkSelectNetwork

TelNwkGetSignalLevel

Purpose Retrieve the selected network carrier signal level.

Declared In TelephonyMgr.h

Prototype Err TelNwkGetSignalLevel(UInt16 iRefnum, TelAppID iAppId, UInt8* oSignalP, UInt16* ioTransIdP)

Parameters

- iRefnum The telephony manager library reference number.

- iAppId The telephone application attachment identifier for your application.

- oSignalP A pointer to an unsigned byte value. Upon return, this is an indication of the signal level in decibels per milliowatt (dBm). The values are explained in the Comments section.

- ioTransIdP Set the value of this parameter to NULL to cause the function to execute synchronously.

    If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns errNone if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the TelEventType event that is sent when the operation completes:
returnCode: errNone upon success or an error code upon failure.

transId: The transaction ID of the operation.

paramP: Points to the unsigned integer value passed to this function in the oSignalP parameter.

functionId: kTelNwkGetSignalLevelMessage

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by oSignalP remains in memory until the asynchronous call completes.

**Comments**
This function sets the value of the variable referenced by oSignalP to an integer value that indicates the signal strength in dBm. The following table describes the signal strength values.

<table>
<thead>
<tr>
<th>Signal level value</th>
<th>dBm value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>≤ 113 dBm</td>
</tr>
<tr>
<td>1</td>
<td>111 dBm</td>
</tr>
<tr>
<td>2 to 30</td>
<td>109 dBm to 53 dBm</td>
</tr>
<tr>
<td>31</td>
<td>≥ 51 dBm</td>
</tr>
<tr>
<td>99</td>
<td>unknown or undetectable</td>
</tr>
</tbody>
</table>

Before using this function, you should verify that it is available by calling the TelIsNwkServiceAvailable macro.

**Compatibility**
 Implemented only if 4.0 New Feature Set is present.
TelNwkSelectNetwork

**Purpose**
Select a network to use from among the set of registered networks.

**Declared In**
TelephonyMgr.h

**Prototype**
```
Err TelNwkSelectNetwork(UInt16 iRefnum, TelAppID iAppId, UInt32 iNetworkId, UInt16* ioTransIdP)
```

**Parameters**
- `iRefnum` The telephony manager library reference number.
- `iAppId` The telephone application attachment identifier for your application.
- `iNetworkId` The identifier of the network to select.
- `ioTransIdP` Set the value of this parameter to `NULL` to cause the function to execute synchronously. If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode` `errNone` upon success or an error code upon failure.
- `transId` The transaction ID of the operation.
- `paramP` Points to the network unsigned integer value passed to this function in the `iNetworkId` parameter.
- `functionId` `kTelNwkSelectNetworkMessage`
Before using this function, you should verify that it is available by calling the `TelIsNwkServiceAvailable` macro.

Implemented only if 4.0 New Feature Set is present.

**See Also**  
TelNwkGetNetworkName, TelNwkGetNetworks, TelNwkGetSelectedNetwork

**TelNwkSetSearchMode**

Sets the search mode used to find a network.

Declared In   
TelephonyMgr.h

Prototype   
`Err TelNwkSetSearchMode(UInt16 iRefnum, 
TelAppID iAppId, UInt8 iMode, UInt16* ioTransIdP)`

Parameters   
- `<-> iRefnum` The telephony manager library reference number.
- `<-> iAppId` The telephone application attachment identifier for your application.
- `<-> iMode` The search mode to use. This must be one of the Network Search Mode Constants.
- `<-> ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result   
Returns errNone if the function was successful or returns an error code if not successful.

Asynchronous Result   
The following fields are updated in the `TelEventType` event that is sent when the operation completes:
returnCode    errNone upon success or an error code upon failure.

tranId        The transaction ID of the operation.

paramP        Points to the unsigned integer value passed to this function in the iMode parameter.

functionId    kTelNwkSetSearchModeMessage

**Comments**    Before using this function, you should verify that it is available by calling the [TelIsNwkServiceAvailable](#) macro.

**Compatibility**    Implemented only if [4.0 New Feature Set](#) is present.

**See Also**    [TelNwkGetSearchMode](#)
Telephony Calls

This chapter describes the telephony calls service sets of the telephony API, which include:

- The data calls service set
- The emergency calls service set
- The speech (voice) calls service set

For more information about the telephony manager basic services and the different service sets, see Chapter 68, “Telephony Basic Services.”

This chapter describes:

- Telephony Calls Data Structures
- Telephony Calls Functions

For more information about using the telephony manager, see Chapter 10, “Telephony Manager,” in Palm OS Programmer’s Companion, vol. II, Communications.

Telephony Calls Data Structures

This section describes the data structures used with the telephony calls service sets portion of the telephony manager API.

TelDtcCallNumberType

The TelDtcCallNumber function uses the TelDataCallNumberType structure to specify information about the telephone number to call.

```c
typedef struct _TelDtcCallNumberType
{
    char    *dialNumberP;
    Uint8   lineId;
} TelDtcCallNumberType
```
Field Descriptions

- `dialNumberP`  The telephone number to dial.

- `lineId`  Upon return from the `TelDtcCallNumber` function, this is the ID of the data line that was established for the telephone call.

TelDtcReceiveDataType

The `TelDtcReceiveData` function uses a `TelDtcReceiveDataType` structure to receive data from an open data line.

```c
typedef struct _TelDtcReceiveDataType
{
    UInt8       *data;
    UInt32      size;
    UInt32      timeout;
} TelDtcReceiveDataType;
```

Field Descriptions

- `data`  A buffer into which the data is stored. Note that if this buffer is too small to contain all of the available data, the end of the data is truncated and the function using this structure generates a `telErrBufferSize` error.

- `size`  When the structure is used as an input parameter, this is the allocated size, in bytes, of the `data` buffer. Upon return, this is the actual number of bytes of data that was retrieved. If the `data` buffer is too small to contain all of the retrieved data, the function using this structure generates a `telErrBufferSize` error.

- `timeout`  The number of milliseconds to wait before timing out.
**TelDtcSendDataType**

The `TelDtcReceiveData` function uses a `TelDtcSendDataType` structure to send data to an open data line.

```c
typedef struct _TelDtcSendDataType
{
    UInt8     *data;
    UInt32    size;
} TelDtcSendDataType
```

**Field Descriptions**

- `data` A pointer to the data to send.
- `size` The number of bytes of data in the data buffer.

**TelEmcGetNumberType**

The `TelEmcGetNumber` function uses a `TelEmcGetNumberType` structure to retrieve an emergency dial telephone number.

```c
typedef struct _TelEmcGetNumberType
{
    UInt8     index;
    UInt8     size;
    Char      *value;
} TelEmcGetNumberType
```

**Field Descriptions**

- `index` The index of the telephone number. This is a zero-based index.
TelEmcSetNumberType

The TelEmcSetNumber function uses a TelEmcNumberType structure to set an emergency dial telephone number.

```c
typedef struct _TelEmcSetNumberType {
    UInt8   index;
    Char    *value;
} TelEmcSetNumberType
```

**Field Descriptions**

- `index`  
  The index of the telephone number. This is a zero-based index.

- `value`  
  The string value of the number to store as the indexth entry.

TelSpcGetCallerNumberType

The TelSpcGetCallerNumber function uses a TelSpcGetCallerNumberType structure to retrieve an incoming telephone number.

<->  size  
When the structure is used as an input parameter, this is the allocated size, in bytes, of the value buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the value buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a telErrBufferSize error.

<->  value  
A null-terminated string buffer into which the emergency dial telephone number is stored.

Note that if this buffer is too small to contain all of the available data, the end of the data is truncated and the function using this structure generates a telErrBufferSize error.
typedef struct _TelSpcGetCallerNumberType
{
    Char    *value;
    UInt8   size;
} TelSpcGetCallerNumberType

Field Descriptions

<-  value  A null-terminated string buffer into which the caller telephone number is stored.

Note that if this buffer is too small to contain all of the available data, the end of the data is truncated and the function using this structure generates a telErrBufferSize error.

<-> size  When the structure is used as an input parameter, this is the allocated size, in bytes, of the value buffer.

Upon return, this is the actual size of the caller dial telephone number, including the null terminator character. If the value buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a telErrBufferSize error.

TelSpcPlayDTMFType

The TelSpcPlayDTMF function uses a TelSpcPlayDTMFType structure to specify the qualities of the DTMF (dual-tone, multi-frequency) sound sent by the phone to the network or remote, connected equipment.

typedef struct _TelSpcPlayDTMFType
    UINT8    keyTone;
    UINT32   duration;
} TelSpcPlayDTMFType
Field Descriptions

-> keyTone  The keycode of the key tone to play. This must be one of the [Keycode Constants](#).

-> duration  The duration of the tone, specified as a multiple of ten milliseconds.

### Telephony Calls Functions

This section describes the functions used with the telephony calls service sets portion of the telephony API.

#### TelDtcCallNumber

**Purpose**  Initiate a data telephone call.

**Declared In**  TelephonyMgr.h

**Prototype**  
```
Err TelDtcCallNumber(UInt16 iRefnum,
TelAppID iAppId, TelDtcCallNumberType *ioParamP,
UInt16 *ioTransIdP)
```

**Parameters**

-> iRefnum  The telephony manager library reference number.

-> iAppId  The telephone application attachment identifier for your application.

<-> ioParamP  A pointer to a [TelDtcCallNumberType](#) structure that specifies information about the telephone call.

<-> ioTransIdP  Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode`: `errNone` upon success or an error code upon failure.
- `transId`: The transaction ID of the operation.
- `paramP`: Points to the `TelDtcCallNumberType` structure passed to this function in the `ioDataCallParamP` parameter.
- `functionId`: `kTelDtcCallNumberMessage`

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioDataCallParamP` remains in memory until the asynchronous call completes.

Comments

Call this function to start a data telephone call. Before using this function, you should verify that it is available by calling the `TelIsDtcServiceAvailable` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelDtcCloseLine`
Telephony Calls
Telephony Calls Functions

### TelDtcCloseLine

**Purpose**
Hang up a data line.

**Declared In**
TelephonyMgr.h

**Prototype**

```
Err TelDtcCloseLine(UInt16 iRefnum, 
TelAppID iAppId, UInt8 iLineId, 
UInt16 *ioTransIdP)
```

**Parameters**

- `iRefnum` The telephony manager library reference number.
- `iAppId` The telephone application attachment identifier for your application.
- `iLineId` The ID of the line to hang up. This is the ID returned by a previous call to the `TelDtcCallNumber` function.
- `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**

Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode` `errNone` upon success or an error code upon failure.
- `transId` The transaction ID of the operation.
- `paramP` Points to the unsigned integer value passed to this function in the `iLineId` parameter.
- `functionId` `kTelDtcCloseLineMessage`
Comments Call this function to end a data telephone call.

Before using this function, you should verify that it is available by calling the `TelIsDtcServiceAvailable` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelDtcCallNumber`

**TelDtcReceiveData**

Purpose Receive data on an opened data communications line.

Declared In `TelephonyMgr.h`

Prototype

```c
Err TelDtcReceiveData(UInt16 iRefnum, 
TelAppID iAppId, TelDtcReceiveDataType *ioParamP, 
UInt16 *ioTransIdP)
```

Parameters

- `iRefnum` The telephony manager library reference number.
- `iAppId` The telephone application attachment identifier for your application.
- `ioParamP` A pointer to a `TelDtcReceiveDataType` structure.
- `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:
**TelDtcSendData**

**Purpose** Send data on an opened data line.

**Declared In** TelephonyMgr.h

**Prototype**

```c
Err TelDtcSendData(UInt16 iRefnum,
                  TelAppID iAppId, TelDtcSendDataType *iParamP,
                  UInt16 *ioTransIdP)
```

**Parameters**

- `-> iRefnum` The telephony manager library reference number.

**Return Values**

- `returnCode` errNone upon success or an error code upon failure.

- `transId` The transaction ID of the operation.

- `paramP` Points to the `TelDtcReceiveDataType` structure passed to this function in the `ioRcvDataP` parameter.

- `functionId` kTelDtcReceiveDataMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

**Comments**

Call this function to receive data during an active data telephone call.

Before using this function, you should verify that it is available by calling the `TelIsDtcServiceAvailable` macro.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.

**See Also**

TelDtcCallNumber, TelDtcCloseLine, TelDtcSendData
Telephony Calls
Telephony Calls Functions

-> iAppId
The telephone application attachment identifier for your application.

-> iParamP
A pointer to a TelDtcSendDataType structure that specifies the data to send.

<-> ioTransIdP
Set the value of this parameter to NULL to cause the function to execute synchronously.
If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the TelEventType event that is sent when the operation completes:

- returnCode
  errNone upon success or an error code upon failure.

- transId
  The transaction ID of the operation.

- paramP
  Points to the TelDtcSendDataType structure passed to this function in the iSendDataP parameter.

- functionId
  kTelDtcSendDataMessage

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by iParamP remains in memory until the asynchronous call completes.

**Comments**
Call this function to send data during an active data telephone call. Before using this function, you should verify that it is available by calling the TelIsDtcServiceAvailable macro.
**Compatibility**

Implemented only if 4.0 New Feature Set is present.

**See Also**

TelDtcCallNumber, TelDtcCloseLine, TelDtcReceiveData

---

**TelEmcCall**

**Purpose**

Call the currently selected emergency service.

**Declared In**

TelephonyMgr.h

**Prototype**

Err TelEmcCall(UInt16 iRefnum, TelAppID iAppId, UInt16 *ioTransIdP)

**Parameters**

- `iRefnum` The telephony manager library reference number.
- `iAppId` The telephone application attachment identifier for your application.
- `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously.
  
  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**

Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**

The following fields are updated in the TelEventType event that is sent when the operation completes:

- `returnCode` errNone upon success or an error code upon failure.
- `transId` The transaction ID of the operation.
- `paramP` A NULL pointer.
- `functionId` kTelEmcCallMessage
Comments

This function calls the telephone number specified in a previous call to the `TelEmcSelectNumber` function. In synchronous mode, this function returns after the dial command has been sent to the phone.

After calling this function, sub-launched applications can receive notifications when the following telephony events occur. Note that these notifications can be raised after both synchronous and asynchronous calls to this function.

Before using this function, you should verify that it is available by calling the `TelIsEmcServiceAvailable` macro.

### Compatibility

Implemented only if [4.0 New Feature Set](#) is present.

### See Also

`TelEmcCloseLine`,
**TelEmcCloseLine**

**Purpose**
Close the line that is currently opened for an emergency telephone call.

**Declared In**
TelephonyMgr.h

**Prototype**
```
Err TelEmcCloseLine(UInt16 iRefnum,
TelAppID iAppId, UInt16 *ioTransIdP)
```

**Parameters**
- `-> iRefnum` The telephony manager library reference number.
- `-> iAppId` The telephone application attachment identifier for your application.
- `<-> ioTransIdP` Set the value of this parameter to `NULL` to cause the function to execute synchronously.
  
  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode` `errNone` upon success or an error code upon failure.
- `transId` The transaction ID of the operation.
- `paramP` A `NULL` pointer.
- `functionId` `kTelDtcCloseLineMessage`

**Comments**
Call this function to end an emergency telephone call.

Before using this function, you should verify that it is available by calling the `TelIsEmcServiceAvailable` macro.
**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
TelEmcCall

---

**TelEmcGetNumber**

**Purpose**
Retrieve an emergency dial telephone number.

**Declared In**
TelephonyMgr.h

**Prototype**
```
Err TelEmcGetNumber(UInt16 iRefnum,
TelAppID iAppId, TelEmcGetNumberType *ioParamP,
UInt16 *ioTransIdP)
```

**Parameters**
- `-> iRefnum` The telephony manager library reference number.
- `-> iAppId` The telephone application attachment identifier for your application.
- `<-> ioParamP` A pointer to a TelEmcGetNumberType structure in which you assign the index of the telephone number that you want to retrieve.

  On input, the size field of this structure specifies the allocated size of the value buffer. Upon return, the size field specifies the actual size of the telephone number, even if it was truncated to fit into the buffer.

- `<-> ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously.

  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns errNone if the function was successful or returns an error code if not successful.
Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- **returnCode**: `errNone` upon success or an error code upon failure.
- **transId**: The transaction ID of the operation.
- **paramP**: Points to the `TelEmcGetNumberType` structure passed to this function in the `ioGetNumberP` parameter.
- **functionId**: `kTelEmcGetNumberMessage`

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

Comments

The emergency call telephone number is stored into the `value` field of the `TelEmcGetNumberType` structure referenced by `ioGetNumberP`. If the `value` buffer is too small to contain the complete string, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The `size` field of the structure is always updated to contain the actual size of the complete string.

Before using this function, you should verify that it is available by calling the `TelIsEmcServiceAvailable` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelEmcGetNumberCount`, `TelEmcSetNumber`, `TelEmcSelectNumber`
**TelEmcGetNumberCount**

**Purpose**
Retrieve the count of emergency dial telephone numbers.

**Declared In**
TelephonyMgr.h

**Prototype**
```c
Err TelEmcGetNumberCount(UInt16 iRefnum, 
TelAppID iAppId, UInt8 *oCountP, 
UInt16 *ioTransIdP)
```

**Parameters**
- `iRefnum` The telephony manager library reference number.
- `iAppId` The telephone application attachment identifier for your application.
- `oCountP` Upon return, the total number of emergency call numbers available.
- `ioTransIdP` Set the value of this parameter to `NULL` to cause the function to execute synchronously. If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode` `errNone` upon success or an error code upon failure.
- `transId` The transaction ID of the operation.
- `paramP` Points to unsigned integer passed to this function in the `oCountP` parameter.
- `functionId` `kTelEmcGetNumberMessage`
**WARNING!** When using this function asynchronously, you must ensure that the value referenced by oCountP remains in memory until the asynchronous call completes.

**Comments**

The emergency telephone call number is stored into the value field of the TelEmcGetNumberType structure referenced by ioGetNumberP. If the value buffer is too small to contain the complete string, the string is truncated (and ends with the null terminator character) and this function returns the telErrBufferSize error. The size field of the structure is always updated to contain the actual size of the complete string.

Before using this function, you should verify that it is available by calling the TelIsEmcServiceAvailable macro.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.

**See Also**

TelEmcGetNumber, TelEmcSetNumber, TelEmcSelectNumber

---

**TelEmcSelectNumber**

**Purpose**

Select the current emergency telephone number. This is the telephone number that gets dialed when you call the TelEmcCall function.

**Declared In**

TelephonyMgr.h

**Prototype**

Err TelEmcSelectNumber(UInt16 iRefnum, TelAppID iAppId, UInt8 iIndex, UInt16 *ioTransIdP)

**Parameters**

- `-> iRefnum` The telephony manager library reference number.
- `-> iAppId` The telephone application attachment identifier for your application.
- `-> iIndex` The zero-based index of the emergency telephone number that you want selected.
<- ioTransIdP  Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**

Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode`  `errNone` upon success or an error code upon failure.
- `transId`  The transaction ID of the operation.
- `paramP`  Points to the unsigned integer value passed to this function in the `iIndex` parameter.
- `functionId`  `kTelEmcSelectNumberMessage`

**Comments**

Before using this function, you should verify that it is available by calling the `TelIsEmcServiceAvailable` macro.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.

**See Also**

`TelEmcCall, TelEmcGetNumber, TelEmcGetNumberCount, TelEmcSetNumber`
**TelEmcSetNumber**

**Purpose**  
Set the telephone number for the specified emergency dial number.

**Declared In**  
TelephonyMgr.h

**Prototype**  
```
Err TelEmcSetNumber(UInt16 iRefnum,
TelAppID iAppId, TelEmcSetNumberType *iParamP,
UInt16 *ioTransIdP)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-&gt; iRefnum</td>
<td>The telephony manager library reference number.</td>
</tr>
<tr>
<td>-&gt; iAppId</td>
<td>The telephone application attachment identifier for your application.</td>
</tr>
<tr>
<td>-&gt; iParamP</td>
<td>A pointer to a <code>TelEmcSetNumberType</code> structure that specifies the telephone number.</td>
</tr>
<tr>
<td>&lt;-&gt; ioTransIdP</td>
<td>Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</td>
</tr>
</tbody>
</table>

**Synchronous Result**  
Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**  
The following fields are updated in the `TelEventTyp` event that is sent when the operation completes:

- `returnCode`  
  errNone upon success or an error code upon failure.

- `transId`  
The transaction ID of the operation.

- `paramP`  
  Points to the `TelEmcSetNumberType` structure passed to this function in the `iNumberP` parameter.

- `functionId`  
kTelEmcSetNumberMessage
WARNING! When using this function asynchronously, you must ensure that the structure referenced by iParamP remains in memory until the asynchronous call completes.

Comments Call this function to associate a new telephone number with the emergency dial number that has the specified iIndex. Before using this function, you should verify that it is available by calling the TelIsEmcServiceAvailable macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also TelEmcGetNumber,

**TelSpcAcceptCall**

Purpose Accept an incoming voice telephone call.

Declared In TelephonyMgr.h

Prototype Err TelSpcAcceptCall(UInt16 iRefnum, TelAppID iAppId, UInt8 *oLineIdP, UInt16 *ioTransIdP)

Parameters

- **-> iRefnum** The telephony manager library reference number.
- **-> iAppId** The telephone application attachment identifier for your application.
- **<- oLineIdP** A pointer to an unsigned byte value. Upon return, this is the ID of the voice line assigned to the telephone call.
- **<- ioTransIdP** Set the value of this parameter to NULL to cause the function to execute synchronously.
If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**

Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode` (errNone upon success or an error code upon failure.)
- `transId` (The transaction ID of the operation.)
- `paramP` (Points to the unsigned integer passed to this function in the `oLineIdP` parameter.)
- `functionId` (kTelSpcAcceptCallMessage)

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by `oLineIdP` remains in memory until the asynchronous call completes.

**Comments**

If another line was active prior to the execution of this function, that line is put on hold. Note that there can only be one line active at any given time, and there can only be one line on hold at any given time.

Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable` macro.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.

**See Also**

TelSpcGetCallerNumber, TelSpcRejectCall
**TelSpcCallNumber**

**Purpose**
Initiate a voice telephone call.

**Declared In**
TelephonyMgr.h

**Prototype**
Err TelSpcCallNumber(UInt16 iRefnum, TelAppID iAppId, const Char *iDialNumberP, UInt16 *ioTransIdP)

**Parameters**
- **iRefnum**
The telephony manager library reference number.
- **iAppId**
The telephone application attachment identifier for your application.
- **iDialNumberP**
A pointer to the telephone number to call.
- **ioTransIdP**
Set the value of this parameter to NULL to cause the function to execute synchronously.
  
  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the **TelEventType** event that is sent when the operation completes:

- **returnCode**
  errNone upon success or an error code upon failure.
- **transId**
The transaction ID of the operation.
- **paramP**
  Points to the string passed to this function in the **iDialNumberP** parameter.
- **functionId**
kTelSpcCallNumberMessage
**WARNING!** When using this function asynchronously, you must ensure that the value referenced by `iDialNumberP` remains in memory until the asynchronous call completes.

**Comments**

A successful return from a synchronous call or receipt of a successful notification from an asynchronous call does not mean that the telephone call has been connected; instead, it indicates that the dial command was sent to the phone. Successful connection of the telephone call is indicated with a sub-launch.

The dial number is formatted according to the following syntax:

\[
\text{DialNumber ::= } \langle \text{Phone\_Number} \rangle \mid \langle \text{Code\_String} \rangle \\
\mid \langle \text{Phone\_Number} \rangle \langle \text{Code\_String} \rangle
\]

\[
\text{Phone\_Number ::= } \langle \text{IntlPrefix}\rangle\langle \text{NatlNumber} \rangle \\
\mid \langle \text{NatlNumber} \rangle
\]

\[
\text{IntlPrefix ::= ' + <country code>}
\]

\[
\text{NatlNumber ::= } \{\text{Pause}<\text{Pause}>\}\{<\text{Digit}>\}
\]

\[
\text{Code\_String ::= } <\text{Symbol}>\{<\text{Symbol}>\}
\]

\[
\text{Symbol ::= } <\text{Digit}> \mid '#' \mid '*'
\]

\[
\text{Digit ::= } '0' \mid '1' \mid '2' \mid '3' \mid '4' \\
\mid '5' \mid '6' \mid '7' \mid '8' \mid '9'
\]

\[
\text{Pause ::= } ','
\]

After calling this function, sub-launched applications can receive notifications when the following telephony events occur. Note that these notifications can be raised after both synchronous and asynchronous calls to this function.
Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable` macro.

**Compatibility**
Implemented only if [4.0 New Feature Set](#) is present.

**See Also**
[TelSpcCloseLine](#)

### TelSpcCloseLine

**Purpose**
Ends a voice telephone call.

**Declared In**
TelephonyMgr.h

**Prototype**

```
Err TelSpcCloseLine(UInt16 iRefnum,
                    TelAppID iAppId, UInt8 iLineId,
                    UInt16 *ioTransIdP)
```

**Parameters**

- `iRefnum` The telephony manager library reference number.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sysTelSpcLaunchCmdCallReleased</code></td>
<td>This is passed to a sub-launched application to warn that the telephone call has been released.</td>
</tr>
<tr>
<td><code>sysTelSpcLaunchCmdCallBusy</code></td>
<td>This is passed to a sub-launched application to warn that the called equipment is busy.</td>
</tr>
<tr>
<td><code>sysTelSpcLaunchCmdCallConnect</code></td>
<td>This is passed to a sub-launched application to warn that the line is open. The ID of the open line is stored in the UInt32 value of the parameter block passed to the application.</td>
</tr>
<tr>
<td><code>sysTelSpcLaunchCmdCallError</code></td>
<td>This is passed to a sub-launched application to warn that the telephone call encountered an error.</td>
</tr>
</tbody>
</table>
Telephony Calls
Telephony Calls Functions

-> iAppId
   The telephone application attachment identifier for your application.

-> iLineId
   The ID of the voice line that you want to close. This is the ID returned by a previous call to the TelSpcAcceptCall function.

<-> ioTransIdP
   Set the value of this parameter to NULL to cause the function to execute synchronously.
   If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result
Returns errNone if the function was successful or returns an error code if not successful.

Asynchronous Result
The following fields are updated in theTelEventType event that is sent when the operation completes:

returnCode errNone upon success or an error code upon failure.

transId The transaction ID of the operation.

paramP Points to the unsigned integer value passed to this function in the iLineId parameter.

functionId kTelSpcCloseLineMessage

Comments
Before using this function, you should verify that it is available by calling theTelIsSpcServiceAvailable macro.

Compatibility
Implemented only if 4.0 New Feature Set is present.

See Also
TelSpcCallNumber
TelSpcConference

**Purpose**
Initiate a conference telephone call by merging the active line and the held line.

**Declared In**
TelephonyMgr.h

**Prototype**
```c
Err TelSpcConference(UInt16 iRefnum, TelAppID iAppId, UInt8 *oLineIdP, UInt16 *ioTransIdP)
```

**Parameters**
- `iRefnum` The telephony manager library reference number.
- `iAppId` The telephone application attachment identifier for your application.
- `oLineIdP` A pointer to an unsigned byte value. Upon return, this is the ID of the voice line assigned to the telephone call.
- `ioTransIdP` Set the value of this parameter to `NULL` to cause the function to execute synchronously. If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode` `errNone` upon success or an error code upon failure.
- `transId` The transaction ID of the operation.
**WARNING!** When using this function asynchronously, you must ensure that the value referenced by oLineIdP remains in memory until the asynchronous call completes.

**Comments**
Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable` macro.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
`TelSpcCallNumber`, `TelSpcCloseLine`, `TelSpcHoldLine`, `TelSpcRetrieveHeldLine`, `TelSpcSelectLine`

---

**TelSpcGetCallerNumber**

**Purpose**
Retrieve the telephone number of the caller on an incoming telephone call.

**Declared In**
`TelephonyMgr.h`

**Prototype**
```c
Err TelSpcGetCallerNumber(UInt16 iRefnum, TelAppID iAppId, TelSpcGetCallerNumberType *ioParamP, UInt16 *ioTransIdP)
```

**Parameters**
- `iRefnum` The telephony manager library reference number.
- `iAppId` The telephone application attachment identifier for your application.
**Telephony Calls**

**Telephony Calls Functions**

<- ioParamP  A pointer to a `TelSpcGetCallerNumberType` structure that is used to retrieve the caller’s telephone number.

On input, the size field of this structure specifies the allocated size of the value buffer. Upon return, the size field specifies the actual size of the telephone number, even if it was truncated to fit into the buffer.

<- ioTransIdP  Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

### Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful. If there is no active incoming telephone call, this function returns the `telErrUnavailableValue` error.

### Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- **returnCode**  `errNone` upon success or an error code upon failure.
- **transId**  The transaction ID of the operation.
- **paramP**  Points to the `TelSpcGetCallerNumberType` structure passed to this function in the `ioParamP` parameter.
- **functionId**  `kTelSpcGetCallerNumberMessage`

**WARNING!**  When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.
Comments
The emergency telephone call number is stored into the value field of the `TelSpcGetCallerNumberType` structure referenced by `ioParamP`. If the value buffer is too small to contain the complete string, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The size field of the structure is always updated to contain the actual size of the complete string.

Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable` macro.

Compatibility
Implemented only if 4.0 New Feature Set is present.

See Also
`TelSpcAcceptCall`

**TelSpcHoldLine**

Purpose
Put the currently active voice line on hold.

Declared In
TelephonyMgr.h

Prototype
Err TelSpcHoldLine (UInt16 iRefnum, TelAppID iAppId, UInt16 *ioTransIdP)

Parameters
- `iRefnum`  The telephony manager library reference number.
- `iAppId`  The telephone application attachment identifier for your application.
- `ioTransIdP`  Set the value of this parameter to NULL to cause the function to execute synchronously.
  
  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result
Returns `errNone` if the function was successful or returns an error code if not successful.
Asynchronous Result

The following fields are updated in the TelEventType event that is sent when the operation completes:

- **returnCode**: errNone upon success or an error code upon failure.
- **transId**: The transaction ID of the operation.
- **paramP**: A NULL pointer.
- **functionId**: kTelSpcHoldLineMessage

Comments

Note that there can only be one line active at any given time, and there can only be one line on hold at any given time.

Before using this function, you should verify that it is available by calling the TelIsSpcServiceAvailable macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

TelSpcRetrieveHeldLine

TelSpcPlayDTMF

**Purpose**

Play a dual-tone multi-frequency sound to the network system for a specified duration. Note that you can only play a DTMF while a voice telephone call is active.

Declared In

TelephonyMgr.h

Prototype

Err TelSpcPlayDTMF(UInt16 iRefnum, TelAppID iAppId, TelSpcPlayDTMFType *iParamP, UInt16 *ioTransIdP)

Parameters

- **iRefnum**: The telephony manager library reference number.
- **iAppId**: The telephone application attachment identifier for your application.
Telephony Calls
Telephony Calls Functions

-> iParamP  A pointer to a TelSpcPlayDTMFType structure that specifies the tone to play and its duration.

<-> ioTransIdP  Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**

Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**

The following fields are updated in the TelEventTypeDef event that is sent when the operation completes:

- **returnCode**  errNone upon success or an error code upon failure.
- **transId**  The transaction ID of the operation.
- **paramP**  Points to the TelSpcPlayDTMFType structure passed to this function in the iParamP parameter.
- **functionId**  kTelSpcPlayDTMFMMessage

**WARNING!**  When using this function asynchronously, you must ensure that the structure referenced by iParamP remains in memory until the asynchronous call completes.

**Comments**

Before using this function, you should verify that it is available by calling the TelIsSpcServiceAvailable macro.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.

**See Also**

TelSpcSendBurstDTMF, TelSpcStartContinuousDTMF, TelSpcStopContinuousDTMF
**TelSpcRejectCall**

**Purpose**
Reject an incoming voice telephone call.

**Declared In**
TelephonyMgr.h

**Prototype**
Err TelSpcRejectCall(UInt16 iRefnum, TelAppID iAppId, UInt16 *ioTransIdP)

**Parameters**
- iRefnum: The telephony manager library reference number.
- iAppId: The telephone application attachment identifier for your application.
- ioTransIdP: Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the TelEventType event that is sent when the operation completes:

- returnCode: errNone upon success or an error code upon failure.
- transId: The transaction ID of the operation.
- paramP: A NULL pointer.
- functionId: kTelSpcRejectCallMessage

**Comments**
Before using this function, you should verify that it is available by calling the TelIsSpcServiceAvailable macro.
Telephony Calls
Telephony Calls Functions

Compatibility
Implemented only if *4.0 New Feature Set* is present.

See Also
TelSpcAcceptCall, TelSpcGetCallerNumber

TelSpcRetrieveHeldLine

Purpose
Reconnect the voice line that is currently on hold, making it the active voice line.

Declared In
TelephonyMgr.h

Prototype
Err TelSpcRetrieveHeldLine(UInt16 iRefnum, TelAppID iAppId, UInt16 *ioTransIdP)

Parameters
- iRefnum The telephony manager library reference number.
- iAppId The telephone application attachment identifier for your application.
- ioTransIdP Set the value of this parameter to NULL to cause the function to execute synchronously.
  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result
Returns errNone if the function was successful or returns an error code if not successful.

Asynchronous Result
The following fields are updated in the TelEventType event that is sent when the operation completes:

- returnType errNone upon success or an error code upon failure.
- transId The transaction ID of the operation.
- paramP A NULL pointer.
- functionId kTelSpcRetrieveHeldLineMessage
Comments
Note that there can only be one line active at any given time, and there can only be one line on hold at any given time.

Before using this function, you should verify that it is available by calling the TelIsSpcServiceAvailable macro.

Compatibility
Implemented only if the 4.0 New Feature Set is present.

See Also
TelSpcHoldLine

TelSpcSelectLine

Purpose
Select the specified line ID as the newly active voice line.

Declared In
TelephonyMgr.h

Prototype
Err TelSpcSelectLine(UInt16 iRefnum, TelAppID iAppId, UInt8 iLineId, UInt16 *ioTransIdP)

Parameters
-> iRefnum
The telephony manager library reference number.

-> iAppId
The telephone application attachment identifier for your application.

-> iLineId
The ID of the voice line that you want to activate. This is the ID returned by a previous call to the TelSpcAcceptCall function.

<-> ioTransIdP
Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result
Returns errNone if the function was successful or returns an error code if not successful.
Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode`: `errNone` upon success or an error code upon failure.
- `transId`: The transaction ID of the operation.
- `paramP`: Points to the unsigned integer value passed to this function in the `iLineId` parameter.
- `functionId`: `kTelSpcSelectLineMessage`

Comments

If a line was active previous to completion of this function, that line is put on hold. Note that there can only be one line active at any given time, and there can only be one line on hold at any given time.

Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelSpcConference`

TelSpcSendBurstDTMF

Purpose

Send a string of dual-tone, multi-frequency sounds to the network system. Note that you can only play a DTMF while a voice telephone call is active.

Declared In

`TelephonyMgr.h`

Prototype

```c
Err TelSpcSendBurstDTMF(UInt16 iRefnum,
TelAppID iAppId, const Char *iDTMFStringP,
UInt16 *ioTransIdP)
```

Parameters

- `iRefnum`: The telephony manager library reference number.
- `iAppId`: The telephone application attachment identifier for your application.
-> iDTMFStringP
A null-terminated string of keytone values. Each byte of the string specifies one of the Keycode Constants.

<-> ioTransIdP
Set the value of this parameter to NULL to cause the function to execute synchronously.
If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the TelEventType event that is sent when the operation completes:

- returnCode: errNone upon success or an error code upon failure.
- transId: The transaction ID of the operation.
- paramP: Points to the string passed to this function in the iDTMFStringP parameter.
- functionId: kTelSpcSendBurstDTMFMessage

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by iDTMFStringP remains in memory until the asynchronous call completes.

**Comments**
This function sends a burst string of keytones to the network. Each key tone is played for the default duration defined by the network. Before using this function, you should verify that it is available by calling the TelIsSpcServiceAvailable macro.
**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
TelSpcPlayDTMF, TelSpcStartContinuousDTMF, TelSpcStopContinuousDTMF

---

**TelSpcStartContinuousDTMF**

**Purpose**
Send a continuous dual-tone, multi-frequency sound to the network system. Note that you can only play a DTMF while a voice telephone call is active.

**Declared In**
TelephonyMgr.h

**Prototype**
Err TelSpcStartContinuousDTMF(UInt16 iRefnum, TelAppID iAppId, UInt8 iKeyCode, UInt16 *ioTransIdP)

**Parameters**
- **iRefnum**
The telephony manager library reference number.
- **iAppId**
The telephone application attachment identifier for your application.
- **iKeyCode**
The keycode to send to the network. This must be one of the Keycode Constants.
- **ioTransIdP**
Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the TelEventType event that is sent when the operation completes:
This function sends a key tone to the network system that is played continuously until the `TelSpcStopContinuousDTMF` function executes.

Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable` macro.

### Compatibility

Implemented only if [4.0 New Feature Set](#) is present.

### See Also

`TelSpcPlayDTMF`, `TelSpcSendBurstDTMF`, `TelSpcStopContinuousDTMF`

### TelSpcStopContinuousDTMF

**Purpose**

Stop the continuous playing of a tone that was started by calling the `TelSpcStartContinuousDTMF` function.

**Declared In**

`TelephonyMgr.h`

**Prototype**

```c
Err TelSpcStopContinuousDTMF(UInt16 iRefnum, 
TelAppID iAppId, UInt16 *ioTransIdP)
```

**Parameters**

- `-> iRefnum` The telephony manager library reference number.
- `-> iAppId` The telephone application attachment identifier for your application.
- `<-> ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously.

```
returnCode errNone upon success or an error code upon failure.
transId The transaction ID of the operation.
paramP Points to the unsigned integer value passed to this function in the iKeyCode parameter.
functionId kTelSpcStartContinuousDTMFMessage
```
If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

### Synchronous Result

Returns errNone if the function was successful or returns an error code if not successful.

### Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- **returnCode**: errNone upon success or an error code upon failure.
- **transId**: The transaction ID of the operation.
- **paramP**: A NULL pointer.
- **functionId**: `kTelSpcStopContinuousDTMFMessage`

### Comments

This function stops the continuous playing of the tone that was previously initiated by calling the `TelSpcStartContinuousDTMF` function.

Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable` macro.

### Compatibility

Implemented only if [4.0 New Feature Set](#) is present.

### See Also

- `TelSpcPlayDTMF`
- `TelSpcSendBurstDTMF`
- `TelSpcStartContinuousDTMF`
Telephony SMS

This chapter describes the telephony SMS service set of the telephony API.

For more information about the telephony manager basic services and the different service sets, see Chapter 68, “Telephony Basic Services.”

This chapter describes:

- Telephony SMS Data Structures
- Telephony SMS Constants
- Telephony SMS Functions

For more information about using the telephony manager, see Chapter 10, “Telephony Manager,” in Palm OS Programmer’s Companion, vol. II, Communications.

Telephony SMS Data Structures

This section describes the data structures used with the SMS service set of the telephony API.

TelSmsDateTimeType

Several of the other data structures used with the Telephony SMS functions include an TelSmsDateTimeType structure to store a date and time value.

```c
typedef _TelSmsDateTimeType
    Boolean    absolute;
    UInt32     dateTime;
} TelSmsDateTimeType
```
Field Descriptions

-> absolute  
If true, the dateTime value is a Palm™ absolute time value, which is the number of seconds since 1/1/1904. If false, the dateTime value is relative to the current date and time.

-> dateTime  
The date and time value.

If the absolute field is true, this is expressed as the number of seconds elapsed since 12:00 A.M. on January 1, 1904. This is the format returned by the TimGetSeconds function.

If the absolute field is false, this is expressed as the number of seconds elapsed from the current time.

TelSmsDeleteMessageType

The TelSmsDeleteMessage function uses a TelSmsDeleteMessageType structure to specify the message to be deleted.

typedef struct _TelSmsDeleteMessageType  
  Uint8   messageType;  
  Uint16  index;  
} TelSmsDeleteMessageType

Field Descriptions

-> messageType  
The message type. This is one of the SMS Message Type Constants.

-> index  
The index of the SMS message in the phone’s storage that is to be deleted. Note that the message is deleted from the storage area selected by a call to the TelSmsSelectStorage function.

Note that the index is zero-based.
TelSmsDeliveryAdvancedCDMAType

The TelSmsDeliveryMessageType structure includes a TelSmsDeliveryAdvancedCDMAType structure for CDMA messages.

typedef struct _TelSmsDeliveryAdvancedCDMAType
{
    UInt8            messageType;
    TelSmsDateTimeType validityPeriod;
    UInt8            priority;
    UInt8            privacy;
    Boolean          alertOnDeliveryRequest;
    Boolean          manualAckRequest;
    UInt8            voiceMessageNumber;
    UInt8            callbackNumberSize;
    Char             *callbackNumberAddress;
    UInt8            languageIndicator;
} TelSmsDeliveryAdvancedCDMAType

Field Descriptions

**messageType**  The type of the message. This must be one of the SMS Message Type Constants.

**validityPeriod**  An TelSmsDateTimeType structure that specifies the amount of time for which the message is valid.

**priority**  The message priority. This must be one of the SMS Message Urgency Constants.

**privacy**  The privacy type of the message. This must be one of the SMS Message Privacy Constants.

**alertOnDeliveryRequest**  true if the user is to be alerted upon delivery of this message, and false if not.

**manualAckRequest**  true if a reply is requested from the user, and false if not.

**voiceMessageNumber**
callbackNumberSize  When the structure is used as an input parameter, this is the allocated size, in bytes, of the callbackNumberAddress string. Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a telErrBufferSize error.

callbackNumberAddress  A buffer into which the callback number address string is stored. Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a telErrBufferSize error.

languageIndicator

TelSmsDeliveryAdvancedGSMType

The TelSmsDeliveryMessageType structure includes a TelSmsDeliveryAdvancedGSMType structure for GSM messages.

typedef struct _TelSmsDeliveryAdvancedGSMType
{
    UInt16     protocolId;
    Boolean    replyPath;
    Char       *serviceCenterNumber;
    UInt8      serviceCenterNumberSize;
} TelSmsDeliveryAdvancedGSMType
Field Descriptions

<- protocolId
   The protocol used for this message. This is one of the SMS Message Transport Protocol Constants.

<- replyPath
   If this value is set, then you use the serviceCenterNumber to reply to this message.

   If this value is not set, you use the default service center provided by your network operator.

<-> serviceCenterNumber
   A buffer into which the service center number string is stored.

   Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a telErrBufferSize error.

<-> serviceCenterNumberSize
   When the structure is used as an input parameter, this is the allocated size, in bytes, of the serviceCenterNumber string.

   Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a telErrBufferSize error.

TelSmsDeliveryAdvancedTDMAType

The TelSmsDeliveryMessageType structure includes a TelSmsDeliveryAdvancedTDMAType structure for TDMA messages.

typedef struct _TelSmsDeliveryAdvancedTDMAType
{
    Uint8 messageType;
    TelSmsDateTimeType validityPeriod;
    Uint8 priority;
}
Telephony SMS
Telephony SMS Data Structures

```c
UInt8 privacy;
Boolean alertOnDeliveryRequest;
Boolean manualAckRequest;
UInt8 voiceMessageNumber;
UInt8 callbackNumberSize;
Char *callbackNumberAddress;
UInt8 languageIndicator;
}

Field Descriptions

messageType
The type of the message. This must be one of the SMS Message Type Constants.

validityPeriod
An TelSmsDateTimeType structure that specifies the amount of time for which the message is valid.

priority
The message priority. This must be one of the SMS Message Urgency Constants.

privacy
The privacy type of the message. This must be one of the SMS Message Privacy Constants.

alertOnDeliveryRequest
true if the user is to be alerted upon delivery of this message, and false if not.

manualAckRequest
true if a reply is requested from the user, and false if not.

voiceMessageNumber
When the structure is used as an input parameter, this is the allocated size, in bytes, of the callbackNumberAddress string.

Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a telErrBufferSize error.
callbackNumberAddress  A buffer into which the callback telephone number address string is stored.

Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a telErrBufferSize error.

languageIndicator

**TelSmsDeliveryMessageType**

The [TelSmsReadMessage](#) function uses a TelSmsDeliveryMessageType structure to retrieve information about a delivered message.

```c
typedef struct _TelSmsDeliveryMessageType
{
    UInt16               version;
    UInt16               index;
    UInt32               messageIdentifier;
    TelSmsDateTimeType   timeStamp;
    UInt16               dataSize;
    UInt8                *data;
    UInt8                dataCodingScheme;
    UInt8                originatingAddressSize;
    Char                 *originatingAddress;
    Boolean              otherToReceive;
    Boolean              reportDeliveryIndicator;
    UInt8                standardType;

    union
    {
        TelSmsDeliveryAdvancedGSMTYPE advancedGSM;
        TelSmsDeliveryAdvancedCDMATYPE advancedCDMA;
        TelSmsDeliveryAdvancedTDMATYPE advancedTDMA;
    } advancedParams;
} TelSmsDeliveryMessageType;
```

[TelSmsReadMessage](#)
Telephony SMS
Telephony SMS Data Structures

```c
UInt8 extensionsCount;
TelSmsExtensionType *extensionsP;
} TelSmsDeliveryMessageType
```

Field Descriptions

- `version` The version of the SMS API associated with this message.

- `index` Upon return, the SMS index of the message on the phone. This is a 0-based index.

  This value is used for input only when calling the `TelSmsReadMessage` function to read one SMS at a time.

- `messageIdentifier` The message identifier.

- `timeStamp` The message time stamp. This is a `TelSmsDateTimeType` structure.

- `dataSize` The size of the data buffer.

  When the structure is used as an input parameter, this is the allocated size of the data buffer.

  Upon return, this is the actual size of the message data. If the buffer is too small to contain the entire message, this field is assigned the entire length of the message, and the function using this structure generates a `telErrBufferSize` error.

- `data` A buffer into which the retrieved data is stored.

  Note that if this buffer is too small to contain the entire retrieved message, the end of the message is truncated and the function using this structure generates a `telErrBufferSize` error.

- `dataCodingScheme` The coding scheme used for the data. This is one of the SMS Data Coding Scheme Constants.
<-> originatingAddressSize  The size of the originatingAddress buffer.

When the structure is used as an input parameter, this is the allocated size of the originatingAddress buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a telErrBufferSize error.

<-> originatingAddress  A buffer into which the originating address string is stored.

Note that if this buffer is too small to contain the entire string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a telErrBufferSize error.

<- otherToReceive  Indicates whether there are more messages waiting to be received from the service center to the mobile device.

<- reportDeliveryIndicator  If true, indicates that the originating user has asked the network to send a delivery report.

<- standardType  Indicates which field of the advancedParams union contains the message information. This is one of the Network Type Constants described in Chapter 70, “Telephony Network.”

If this value is kTelNwkCDMA, then the advancedParams union contains a TelSmsDeliveryAdvancedCDMAType structure, and similarly for the other values.
The `TelSmsExtensionType` structure specifies multipart information about a message.

```c
typedef struct _TelSmsExtensionType
    UInt8    extensionTypeId;
union
    {
        TelSmsMultiPartExtensionType    mp;
        TelSmsNbsExtensionType          nbs;
        TelSmsUserExtensionType         user;
    } extension;
} TelSmsExtensionType
```

**advancedParams**
Advanced message information for GSM, CDMA, or TDMA messages. This is one of the following structure types:
- `TelSmsDeliveryAdvancedGSMType`
- `TelSmsDeliveryAdvancedCDMAType`
- `TelSmsDeliveryAdvancedTDMAType`

**extensionsCount**
On input, this is the number of extension structures allocated for this message. You must allocate at least one structure to specify the multipart information.

Upon return, this is the number of extensions in the SMS header. If the SMS header contains more extensions than you have allocated, the available extension structures are filled, and this function generates a `telErrBufferSize` error.

**extensionsP**
A pointer to an array of `TelSmsExtensionType` structures that you have allocated for this message. You must allocate this array before using this structure.
Field Descriptions

extensionTypeId  Identifies the type of SMS extension structure found in the extension union. This is one of the SMS Extension Type Constants.

If the value of this field is kTelSmsMultipartExtensionTypeId, then the extension union contains a TelSmsMultiPartExtensionType structure.

If the value of this field is kTelSmsNbsExtensionTypeId or kTelSmsNbs2ExtensionTypeId, then the union contains a TelSmsNbsExtensionType structure. The difference between these two is that kTelSmsNbs2ExtensionTypeId indicates that the port value is a long instead of a short.

If this field contains any other value, then the union contains a TelSmsUserExtensionType structure.

extension  The extension information, which is one of the following structure types:
TelSmsMultiPartExtensionType
TelSmsNbsExtensionType
TelSmsUserExtensionType

TelSmsGetAvailableStorageType

The TelSmsGetAvailableStorage function uses a TelSmsGetAvailableStorageType structure to retrieve information about the storage available on the phone.

typedef struct _TelSmsGetAvailableStorageType
UInt16  count;
UInt8   *storagesP;
} TelSmsGetAvailableStorageType
Field Descriptions

<-> count  The size of the storagesP buffer. When the structure is used as an input parameter, this is the allocated number of values in the buffer. Upon return, this is the total number of storage areas in the phone.

<-> storagesP  A buffer into which the retrieved storage IDs are stored. Each value stored into the buffer is one of the SMS Storage ID Constants.

TelSmsGetMessageCountType

The TelSmsGetMessageCount function uses a TelSmsGetMessageCountType structure to retrieve information about messages in the currently selected storage.

typedef struct _TelSmsGetMessageCountType
{
    UInt8    messageType;
    UInt16   slots;
    UInt16   count;
} TelSmsGetMessageCountType

Field Descriptions

-> messageType  The type of message for which you want to retrieve the count. This must be one of the SMS Message Type Constants.

You must fill this in on input to the TelSmsGetMessageCount function.

<- slots  The total number of message slots available in the currently selected storage area (for all message types).

<- count  The number of filled slots for the specified messageType in the currently selected storage area.
TelSmsManualAckType

The TelSmsSendManualAcknowledge function uses a TelSmsManualAckType structure to specify the information used to send a message acknowledgment.

```c
typedef struct _TelSmsManualAckType {
    UInt16     version;
    Char       *destinationAddress;
    UInt32     messageId;
    UInt16     dataSize;
    UInt8      *data;
    UInt8      dataCodingScheme;
    UInt8      responseCode;
} TelSmsManualAckType
```

**Field Descriptions**

- **version**
  The version of the SMS API used for this acknowledgment.

- **destinationAddress**
  The destination address. For GSM, the length of this address is 12 bytes. For CDMA, the length is up to 128 bytes.

- **messageId**
  Upon return, the ID of the acknowledgment.

- **dataSize**
  The size of the data buffer.

  When the structure is used as an input parameter, this is the allocated size of the data buffer.

  Upon return, this is the actual size of the data. If the buffer is too small to contain all of the data, this field is assigned the entire length of the data, and the function using this structure generates a telErrBufferSize error.

- **data**
  A buffer into which the retrieved data is stored.

  Note that if this buffer is too small to contain the all of the retrieved data, the data is truncated and the function using this structure generates a telErrBufferSize error.
dataCodingScheme The coding scheme used for the data. This must be one of the SMS Data Coding Scheme Constants.

responseCode The response code. The value of this field depends on the network being used.

**TelSmsMultiPartExtensionType**

The *TelSmsExtensionType* structure uses a *TelSmsMultiPartExtensionType* structure to describe information about a multipart message.

```c
typedef struct _TelSmsMultiPartExtensionType
    UInt16  bytesSent;
    UInt16  partCurrent;
    UInt16  partCount;
    UInt16  partId;
} TelSmsMultiPartExtensionType
```

**Field Descriptions**

**<--> bytesSent**

On input, set this value to 0.

Upon return, this is the current count of message bytes that have been sent.

**<--> partCurrent**

On input, set this value to 0.

Upon return, this is the part number of the current message part.
TelSmsNbsExtensionType

The TelSmsExtensionType structure uses a TelSmsNbsExtensionType structure to describe information about a NBS message.

```c
typedef struct _TelSmsNbsExtensionType
    UInt16  destPort;
    UInt16  srcPort;
} TelSmsNbsExtensionType
```

Field Descriptions

<> partCount

On input, set this value to 0.

Upon return, this is the number of message parts required to send the data.

<> partId

The ID of the current SMS message. This ID is unique and is the same for all parts of the message. This information is required to reassemble a multi-part SMS.

On input, set this value to 0.

TelSmsReadMessagesType

The TelSmsReadMessages function uses a TelSmsReadMessagesType structure to retrieve messages from the currently selected storage area.

```c
typedef struct _TelSmsReadMessagesType
    UInt16                      first;
    UInt16                      count;
    TelSmsDeliveryMessageType  *messagesP;
```
Field Descriptions

-> first
   The index of the first message to retrieve. Message indexes are zero-based.

<-> count
   The size of the messagesP buffer.

When the structure is used as an input parameter, this is the allocated number of pointers in the messagesP buffer.

Upon return, this is the number of messages that could be retrieved. If the messagesP buffer is too small to contain all of the messages, this field is assigned the entire count, and the function using this structure generates a telErrBufferSize error.

<-> messagesP
   An array of pointers to TelSmsDeliveryMessageType structures, each of which is filled in with a retrieved message, if available.

TelSmsReadReportsType

The TelSmsReadReports function uses a TelSmsReadReportsType structure to retrieve reports from the currently selected storage area.

typedef struct _TelSmsReadReportsType

   Uint16             first;
   Uint16             count;
   TelSmsReportType   *reportsP;

} TelSmsReadReportsType
Field Descriptions

-> first
The index of the first report to retrieve. Report indexes are zero-based.

<-> count
The size of the reportsP buffer.

When the structure is used as an input parameter, this is the allocated number of pointers in the reportsP buffer.

Upon return, this is the actual number of reports that could be read. If the buffer is too small to contain all of the reports, this field is assigned the entire count, and the function using this structure generates a telErrBufferSize error.

<-> reportsP
An array of pointers to TelSmsReportType structures, each of which is filled in with a retrieved message, if available.

Note that if this buffer is too small to contain all of the retrieved reports, the function using this structure generates a telErrBufferSize error.

TelSmsReadSubmittedMessagesType
The TelSmsReadSubmittedMessages function uses a TelSmsReadSubmittedMessagesType structure to retrieve submitted messages from the currently selected storage area.

```c
typedef struct _TelSmsReadMessagesType
    UInt16          first;
    UInt16          count;
    TelSmsDeliveryMessageType  *submittedsP;
} TelSmsReadSubmittedMessagesType
```
Field Descriptions

-> first
The index of the first message to retrieve. Message indexes are zero-based.

<-> count
The size of the submittedsP buffer.

When the structure is used as an input parameter, this is the allocated number of pointers in the submittedsP buffer. Upon return, this is the number of messages that were actually read.

<-> submittedsP
An array of pointers to TelSmsSubmittedMessageType structures, each of which is filled in with a retrieved message, if available.

TelSmsReportType

The TelSmsReadReport function uses a TelSmsReportType structure to retrieve a report from the report storage area.

typedef struct _TelSmsReportType
{
    UInt16 version;
    UInt16 index;
    UInt8 reportType;
    UInt32 messageId;
    UInt16 dataSize;
    UInt8 *data;
    UInt8 dataCodingScheme;
    Char *originatingAddress;
    UInt8 originatingAddressSize;
    UInt8 report;
    TelSmsDateTimeType timeStamp;
} TelSmsReportType

Field Descriptions

-> version

<-> index
The index of the report in the phone storage area.

<- reportType
The delivery report type.
<-> messageId
The message ID.
<-> dataSize
The size of the data buffer.

When the structure is used as an input parameter, this is the allocated size of the data buffer.

Upon return, this is the actual size of the data. If the buffer is too small to contain all of the data, this field is assigned the entire length of the data, and the function using this structure generates a telErrBufferSize error.

<-> data
A buffer into which the retrieved data is stored.

Note that if this buffer is too small to contain the all of the retrieved data, the data is truncated and the function using this structure generates a telErrBufferSize error.

<-> dataCodingScheme
The encoding scheme used for the report data. This must be one of the SMS Data Coding Scheme Constants.

<-> originatingAddress
A buffer into which the originating address is stored.

Note that if this buffer is too small to contain the entire retrieved string, the string is truncated (and ends with the null terminator character) and the function using this structure generates a telErrBufferSize error.

<-> originatingAddressSize
The size of the originatingAddress buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a telErrBufferSize error.
TelSmsSendMessageType

typedef struct _TelSmsSendMessageType
{
    UInt32                   messageId;
    TelSmsSubmitMessageType  message;
} TelSmsSendMessageType

Field Descriptions

<- messageId
The SMS ID that was assigned by the telephone to the outgoing message.

For a multi-part message, each part has its own message ID.

-> message
A structure of type TelSmsSubmitMessageType that contains the message data and parameters.

TelSmsSubmitAdvancedCDMAType

The TelSmsSubmitMessage structure includes a TelSmsSubmitAdvancedCDMAType structure for CDMA messages.

typedef struct _TelSmsSubmitAdvancedCDMAType
{
    Boolean              manualAckRequest;
    UInt8                messageType;
    TelSmsDateTimeType   deferredDate;
    UInt8                priority;
    UInt8                privacy;
    Boolean              alertOnDeliveryRequest;
    Char                 *callbackNumber;
    UInt8                callbackNumberSize;
} TelSmsSubmitAdvancedCDMAType
Field Descriptions

manualAckRequest  true if a reply is requested from the user, and false if not.

messageType  The type of the message. This must be one of the SMS Message Type Constants.

defferredDate

priority

privacy  The privacy type of the message. This must be one of the SMS Message Privacy Constants.

alertOnDeliveryRequest  true if the user is to be alerted upon delivery of this message, and false if not.

callbackNumber  A buffer into which the retrieved callback telephone number string is stored.

Note that if this buffer is too small to contain the entire retrieved string, the string is truncated (and ends with the null terminator character) and the function using this structure generates a telErrBufferSize error.

callbackNumberSize  When the structure is used as an input parameter, this is the allocated size, in bytes, of the callbackNumber string.

Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a telErrBufferSize error.

TelSmsSubmitAdvancedGSMType

The TelSmsSubmitMessageType structure includes a TelSmsSubmitAdvancedGSMType structure for CSM messages.
```c
typedef struct __TelSmsSubmitAdvancedGSMTyp{
    UInt16    protocolId;
    Boolean   rejectDuplicatedRequest;
    Boolean   replyPath;
    Char      *serviceCenterNumber;
    UInt8     serviceCenterNumberSize;
} TelSmsSubmitAdvancedGSMTyp
```

**Field Descriptions**

- **protocolId**
  - Specifies gateway information for routing a message to another transport.
  - Some service centers provide a gateway between SMS and other transports such as mail and FAX. Service centers may reject messages with protocolId values that are reserved or unsupported.
  - The mobile device does not interpret reserved or unsupported values, but does store them as received.

- **rejectDuplicatedRequest**
  - A Boolean value that specifies if the service center should accept a submit message for a submit message that is still held in the service center when that message has the same identifier and destination address as a previously submitted message from the same originating address.

- **replyPath**
  - The path that the service center can use to deliver a reply to the originating message.
  - The reply path is requested by the originating mobile device by setting the replyPath parameter in the original submit message.
  - If the service center supports reply path requests from the mobile device, the service center sets the replyPath parameter in the response.
TelSmsSubmitMessageType structure includes a TelSmsSubmitAdvancedTDMAType structure for TDMA messages.

```
typedef struct _TelSmsSubmitAdvancedTDMAType
{
    Boolean              manualAckRequest;
    UInt8                messageType;
    TelSmsDateTimeType   deferredDate;
    UInt8                priority;
    UInt8                privacy;
    Boolean              alertOnDeliveryRequest;
    Char                 *callbackNumber;
    UInt8                callbackNumberSize;
} TelSmsSubmitAdvancedTDMAType
```

**Field Descriptions**

- **manualAckRequest**
  true if a reply is requested from the user, and false if not.

- **messageType**
  The type of the message. This must be one of the SMS Message Type Constants.

- **deferredDate**

- **priority**

- **privacy**
  The privacy type of the message. This must be one of the SMS Message Privacy Constants.

- **alertOnDeliveryRequest**
  true if the user is to be alerted upon delivery of this message, and false if not.
callbackNumber A buffer into which the retrieved callback telephone number string is stored.

Note that if this buffer is too small to contain the entire retrieved string, the string is truncated (and ends with the null terminator character) and the function using this structure generates a telErrBufferSize error.

callbackNumberSize When the structure is used as an input parameter, this is the allocated size, in bytes, of the callbackNumber string.

Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a telErrBufferSize error.

TelSmsSubmitMessageType
The TelSmsReadReports and TelSmsSendMessageType structures use a TelSmsSubmitMessageType structure to stored reports retrieved from the currently selected storage area.

typedef struct _TelSmsSubmitMessageType
{
    UInt16 version;
    Boolean networkDeliveryRequest;
    Char *destinationAddress;
    UInt8 destinationAddressSize;
    UInt16 dataSize;
    UInt8 *data;
    UInt8 dataCodingScheme;
    TelSmsDateTimeType validityPeriod;
    UInt8 standardType;
    union
    {
        TelSmsSubmitAdvancedGSMTypex advancedGSM;
        TelSmsSubmitAdvancedCDMAType advancedCDMA;
    }
}
TelSmsSubmitAdvancedTDMAType  advancedTDMA;
} advancedParams;
UInt8                  extensionsCount;
TelSmsExtensionType    *extensionsP;
} TelSmsSubmitMessageType

Field Descriptions

-> version      The version of the SMS API associated with this message.

-> networkDeliveryRequest If this value is true, the service center accepts the submit message.

<-> destinationAddress A buffer that contains the phone number of the message recipient.

<-> destinationAddressSize The size of the destination address string.

<-> dataSize      The size of the data buffer.

<-> data          A buffer into which the retrieved message data is stored.

-> dataCodingScheme The coding scheme used for the data. This is one of the SMS Data Coding Scheme Constants.

-> validityPeriod An TelSmsDateTimeType structure that specifies the amount of time for which the message is valid.

-> standardType   Indicates which field of the advancedParams union contains the message information. This must be one of the Network Type Constants described in Chapter 70, “Telephony Network.”

If this value is kTelNwkCDMA, then the advancedParams union contains a TelSmsDeliveryAdvancedCDMAType structure, and similarly for the other values.
The TelSmsReadSubmittedMessage function uses a TelSmsSubmittedMessageType structure to retrieve reports from the currently selected storage area.

```
typedef struct _TelSmsSubmittedMessageType
{
    UInt16                  index;
    TelSmsSubmitMessageType  message;
} TelSmsSubmittedMessageType
```

Field Descriptions

- `index` The index of the message on the phone.
- `message` A TelSmsSubmitMessageType structure that represents the message.

TelSmsUserExtensionType

The TelSmsExtensionType structure uses a TelSmsUserExtensionType structure to describe a user-defined extended message header.

```
typedef struct _TelSmsUserExtensionType
    UInt8   *extHeader;
    UInt8   extHeaderSize;
} TelSmsUserExtensionType
```
Telephony SMS
Telephony SMS Constants

Field Descriptions

<-> extHeader
On input, this field must be set to 0.

Upon return, this is a pointer to the user-defined header content.

<-> extHeaderSize
On input, this field must be set to 0.

Upon return, this is the size of the user-defined header content.

Telephony SMS Constants

This section describes the constants used with the SMS service set of the telephony API.

SMS Extension Type Constants

The SMS extension type constants describe the type of extension used to represent part of a message.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTelSmsMultiPartExtensionTypeId</td>
<td>0x00</td>
<td>A multipart short message.</td>
</tr>
<tr>
<td>kTelSmsNbsExtensionTypeId</td>
<td>0x04</td>
<td>An NBS message with short port number value.</td>
</tr>
<tr>
<td>kTelSmsNbs2ExtensionTypeId</td>
<td>0x05</td>
<td>An NBS message with long port number value.</td>
</tr>
<tr>
<td>(any other value)</td>
<td></td>
<td>A user-defined extension type.</td>
</tr>
</tbody>
</table>

SMS Message Type Constants

The SMS message type constants describe the delivery type of a message.
### SMS Message Transport Protocol Constants

The SMS message transport protocol constants describe the protocol used to deliver a message.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTelSmsDefaultProtocol</td>
<td>0</td>
<td>The default message transport protocol.</td>
</tr>
<tr>
<td>kTelSmsFaxProtocol</td>
<td>1</td>
<td>A FAX message.</td>
</tr>
<tr>
<td>kTelSmsX400Protocol</td>
<td>2</td>
<td>An X.400 message.</td>
</tr>
<tr>
<td>kTelSmsPagingProtocol</td>
<td>3</td>
<td>A paged message.</td>
</tr>
<tr>
<td>kTelSmsEmailProtocol</td>
<td>4</td>
<td>An email message.</td>
</tr>
<tr>
<td>kTelSmsErmesProtocol</td>
<td>5</td>
<td>An Ermes message.</td>
</tr>
<tr>
<td>kTelSmsVoiceProtocol</td>
<td>6</td>
<td>A voice message.</td>
</tr>
</tbody>
</table>
SMS Storage ID Constants
The SMS storage ID constants describe the storage location of a message.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTelSmsStorageSIM</td>
<td>0</td>
<td>Stored in the SIM.</td>
</tr>
<tr>
<td>kTelSmsStoragePhone</td>
<td>1</td>
<td>Stored in the phone.</td>
</tr>
<tr>
<td>kTelSmsStorageAdaptor</td>
<td>2</td>
<td>Stored in the telephone adaptor.</td>
</tr>
<tr>
<td>kTelSmsStorageFirstOem</td>
<td>3</td>
<td>Storage managed by the OEM.</td>
</tr>
</tbody>
</table>

This constant specifies the first OEM storage area. You can specify additional OEM storage areas by incrementing this value. For example, to specify the third OEM storage area, use the following:

```
kTelSmsStorageFirstOem+2
```

SMS Data Coding Scheme Constants
The SMS data coding scheme constants describe the encoding used for SMS data.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTelSms8BitsEncoding</td>
<td>0</td>
<td>8-bit encoding.</td>
</tr>
<tr>
<td>kTelSmsBitsASCIIEncoding</td>
<td>1</td>
<td>ANSI X3.4 encoding.</td>
</tr>
<tr>
<td>kTelSmsIA5Encoding</td>
<td>2</td>
<td>CCITT T.50 encoding.</td>
</tr>
<tr>
<td>kTelSmsIS91Encoding</td>
<td>3</td>
<td>TIA/EIA/IS-91 section 3.7.1 encoding.</td>
</tr>
<tr>
<td>kTelSmsUCS2Encoding</td>
<td>4</td>
<td>UCS2 encoding; used with GSM only.</td>
</tr>
<tr>
<td>kTelSmsDefaultGSMEncoding</td>
<td>5</td>
<td>Default encoding for GSM only.</td>
</tr>
</tbody>
</table>
SMS Message Urgency Constants

The SMS message urgency constants describe the priority level of a message in a `TelSmsDeliveryAdvancedCDMAType` or `TelSmsDeliveryAdvancedTDMAType` structure.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTelSmsUrgencyNormal</td>
<td>0</td>
<td>Normal urgency.</td>
</tr>
<tr>
<td>kTelSmsUrgencyUrgent</td>
<td>1</td>
<td>An urgent message.</td>
</tr>
<tr>
<td>kTelSmsUrgencyEmergency</td>
<td>2</td>
<td>An emergency message.</td>
</tr>
</tbody>
</table>

SMS Message Privacy Constants

The SMS message privacy constants describe the privacy type of a message in a CDMA or TDMA advanced parameters.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTelSmsPrivacyNotRestricted</td>
<td>0</td>
<td>Privacy level 0.</td>
</tr>
<tr>
<td>kTelSmsPrivacyRestricted</td>
<td>1</td>
<td>Privacy level 1.</td>
</tr>
<tr>
<td>kTelSmsPrivacyConfidential</td>
<td>2</td>
<td>Privacy level 2.</td>
</tr>
<tr>
<td>kTelSmsPrivacySecret</td>
<td>3</td>
<td>Privacy level 3.</td>
</tr>
</tbody>
</table>

Telephony SMS Functions

This section describes the functions used with the SMS service set of the telephony API.
TelSmsDeleteMessage

**Purpose**  
Delete an SMS report, delivered message, or submitted message.

**Declared In**  
TelephonyMgr.h

**Prototype**  
Err TelSmsDeleteMessage(UInt16 iRefnum, TelAppID iAppId, TelSmsDeleteMessageType *ioParamP, UInt16 *ioTransIdP)

**Parameters**

- **iRefnum**  
The telephony manager library reference number.

- **iAppId**  
The telephone application attachment identifier for your application.

- **ioParamP**  
A pointer to a `TelSmsDeleteMessageType` structure that specifies the index and type of the message to delete.

- **ioTransIdP**  
Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**  
Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**  
The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- **returnCode**  
  errNone upon success or an error code upon failure.

- **transId**  
The transaction ID of the operation.
paramP  Points to the `TelSmsDeleteMessageType` structure passed to this function in the `iDelInfoP` parameter.

functionId  kTelSmsDeleteMessageMessage

**WARNING!**  When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

**Comments**  If the deleted message has been delivered, the deletion is performed in the current storage, which you can set with the `TelSmsSelectStorage` function.

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable` macro.

**Compatibility**  Implemented only if 4.0 New Feature Set is present.

**See Also**  `TelSmsSelectStorage`

### TelSmsGetAvailableStorage

**Purpose**  Retrieve the list of all available storage on the phone.

**Declared In**  TelephonyMgr.h

**Prototype**  
```c
Err TelSmsGetAvailableStorage(UInt16 iRefnum, TelAppID iAppId, TelSmsGetAvailableStorageType *ioParamP, UInt16 *iTransIdP)
```

**Parameters**  
- `-> iRefnum`  The telephony manager library reference number.
- `-> iAppId`  The telephone application attachment identifier for your application.
Telephony SMS
Telephony SMS Functions

<-> ioParamP  A pointer to a TelSmsGetAvailableStorageType structure that is filled in with information about the storage areas available on the phone.

<-> ioTransIdP Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result
Returns errNone if the function was successful or returns an error code if not successful.

Asynchronous Result
The following fields are updated in the TelEventType event that is sent when the operation completes:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>returnCode</td>
<td>errNone upon success or an error code upon failure.</td>
</tr>
<tr>
<td>transId</td>
<td>The transaction ID of the operation.</td>
</tr>
<tr>
<td>paramP</td>
<td>Points to the TelSmsGetAvailableStorageType structure passed to this function in the ioParamP parameter.</td>
</tr>
<tr>
<td>functionId</td>
<td>kTelSmsGetAvailableStorageMessage</td>
</tr>
</tbody>
</table>

WARNING! When using this function asynchronously, you must ensure that the structure referenced by ioParamP remains in memory until the asynchronous call completes.

Comments
The count of storage areas available on the phone is stored into the count field of the TelSmsGetAvailableStorageType structure referenced by ioParamP, and the storage ID of each available room is stored into the buffer referenced by the storagesP field. If the storagesP buffer is too small to contain all of the storage IDs, the buffer is truncated and this function returns the
TelErrBufferSize error. The count field of the structure is always updated to contain the total number of available storage areas on the phone.

Before using this function, you should verify that it is available by calling the TelIsSmsServiceAvailable macro.

Compatibility
Implemented only if 4.0 New Feature Set is present.

See Also TelSmsSelectStorage

TelSmsGetDataMaxSize

Purpose
Returns the maximum length, in bytes, of a message on the current network.

Declared In TelephonyMgr.h

Prototype
Err TelSmsGetDataMaxSize(UInt16 iRefnum, TelAppID iAppId, UInt16 *oSizeP, UInt16 *ioTransIdP)

Parameters
- iRefnum The telephony manager library reference number.
- iAppId The telephone application attachment identifier for your application.
- oSizeP A pointer to an unsigned integer value that is updated with the maximum length of an SMS message on the current network.

- ioTransIdP Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result
Returns errNone if the function was successful or returns an error code if not successful.
### Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- **returnCode**
  - `errNone` upon success or an error code upon failure.

- **transId**
  - The transaction ID of the operation.

- **paramP**
  - Points to the unsigned integer value passed to this function in the `oSizeP` parameter.

- **functionId**
  - `kTelGetDataMaxSizeMessage`

---

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by `oSizeP` remains in memory until the asynchronous call completes.

---

### Comments

You can use this function to determine the maximum size you need to allocate to read a message.

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable` macro.

### Compatibility

Implemented only if [4.0 New Feature Set](#) is present.

### See Also

- [TelSmsReadMessage](#), [TelSmsReadMessages](#)
TelSmsGetMessageCount

Purpose
Retrieve the total number of message slots, and the number of filled slots for the specified message type.

Declared In
TelephonyMgr.h

Prototype
Err TelSmsGetMessageCount(UInt16 iRefnum,
TelAppID iAppId,
TelSmsGetMessageCountType *ioParamP,
UInt16 *ioTransIdP)

Parameters
-> iRefnum The telephony manager library reference number.
-> iAppId The telephone application attachment identifier for your application.
<-> ioParamP A pointer to a TelSmsGetMessageCountType structure that specifies the message type and is filled in with the count information.
<-> ioTransIdP Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result
Returns errNone if the function was successful or returns an error code if not successful.

Asynchronous Result
The following fields are updated in the TelEventType event that is sent when the operation completes:

returnCode errNone upon success or an error code upon failure.

transId The transaction ID of the operation.
**paramP**  
Points to the `TelSmsGetMessageCountType` structure passed to this function in the `ioParamP` parameter.

**functionId**  
kTelSmsGetMessageCountMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

---

**Comments**  
The currently selected storage area pertains only to delivered messages; other message types are stored on the phone, but not in a specific storage area. If you specify delivered messages, this function retrieves information about the messages in the currently selected SMS storage area on the phone. If you specify a different message type, this function retrieves information about the messages in the phone.

You specify the message type by assigning one of the `SMS Message Type Constants` to the `messageType` field of the `TelSmsGetMessageCountType` structure before calling this function.

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable` macro.

**Compatibility**  
Implemented only if 4.0 New Feature Set is present.

**See Also**  
`TelSmsSelectStorage`
**TelSmsGetSelectedStorage**

**Purpose**
Retrieve the ID of the currently selected storage area on the phone.

**Declared In**
TelephonyMgr.h

**Prototype**
```c
Err TelSmsGetSelectedStorage(UInt16 iRefnum, TelAppID iAppId, UInt8 *oStorageIdP, UInt16 *ioTransIdP)
```

**Parameters**
- `-> iRefnum` The telephony manager library reference number.
- `-> iAppId` The telephone application attachment identifier for your application.
- `<- oStorageIdP` A pointer to an unsigned byte value that is assigned the ID of the currently selected storage area on the phone. The assigned ID value is one of the SMS Storage ID Constants.
- `<-> ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the TelEventType event that is sent when the operation completes:

- `returnCode` errNone upon success or an error code upon failure.
- `transId` The transaction ID of the operation.
Parameter Description

paramP Points to the unsigned integer value passed to this function in the oStorageIdP parameter.

functionId kTelGetSelectedStorageMessage

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by oStorageIdP remains in memory until the asynchronous call completes.

**Comments**

The currently selected storage area pertains only to delivered messages; other message types are stored on the phone, but not in a specific storage area.

Before using this function, you should verify that it is available by calling the TelIsSmsServiceAvailable macro.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.

**See Also**

TelSmsSelectStorage

### TelSmsGetUniquePartId

**Purpose**

Return a unique part identifier to assign to the partId field of a submit message.

**Declared In**

TelephonyMgr.h

**Prototype**

Err TelSmsGetUniquePartId(UInt16 iRefnum, TelAppID iAppId, UInt16 *oUniqueIdP, UInt16 *ioTransIdP)

**Parameters**

- **-> iRefnum** The telephony manager library reference number.
- **-> iAppId** The telephone application attachment identifier for your application.
- **<- oUniqueIdP** A pointer to a unsigned integer value. Upon return, this is the unique part ID value.
Telephony SMS
Telephony SMS Functions

<> ioTransIdP  Set the value of this parameter to NULL to cause the function to execute synchronously.
If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the TelEventType event that is sent when the operation completes:

- **returnCode**  errNone upon success or an error code upon failure.
- **transId**  The transaction ID of the operation.
- **paramP**  Points to the unsigned integer value passed to this function in the oUniqueIdP parameter.
- **functionId**  kTelGetUniquePartIdMessage

**WARNING!**  When using this function asynchronously, you must ensure that the value referenced by oUniquePartIdP remains in memory until the asynchronous call completes.

**Comments**
This function corresponds to the kTelUrqSmsGetUniquePartIdMessage value.
Before using this function, you should verify that it is available by calling the TelIsSmsServiceAvailable macro.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
TelSmsSendMessage
TelSmsReadMessage

**Purpose** Retrieve a delivered message from the currently selected storage area.

**Declared In** TelephonyMgr.h

**Prototype**
```
Err TelSmsReadMessage(UInt16 iRefnum,
TelAppID iAppId,
TelSmsDeliveryMessageType *ioMessageP,
UInt16 *ioTransIdP)
```

**Parameters**
- **iRefnum** The telephony manager library reference number.
- **iAppId** The telephone application attachment identifier for your application.
- **ioMessageP** A pointer to a `TelSmsDeliveryMessageType` structure that is filled in with the message.

  On input, the `index` field of this structure contains the index of the message that you want retrieved. Message indexes are zero-based.

  On input, the `extensionsCount` field specifies the number of extensions allocated in the `extensionsP` array. You must allocate at least one multi-part extension, even for a single-part message.

  On input, the `dataSize` field specifies the allocated size of the data buffer, and the `originatingAddressSize` field specifies the allocated size of the originatingAddress string. Upon return, each size field specifies the complete size of the data that was stored into the buffer, even if the data had to be truncated to fit.
Telephony SMS
Telephony SMS Functions

 <-> ioTransIdP  Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

<table>
<thead>
<tr>
<th>Synchronous Result</th>
<th>Asynchronous Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns errNone if the function was successful or returns an error code if not successful.</td>
<td>The following fields are updated in the TelEventType event that is sent when the operation completes:</td>
</tr>
</tbody>
</table>

- **returnCode**  errNone upon success or an error code upon failure.
- **transId**  The transaction ID of the operation.
- **paramP**  Points to the TelSmsDeliveryMessageType structure passed to this function in the ioMessageP parameter.
- **functionId**  kTelSmsReadMessageMessage

**WARNING!**  When using this function asynchronously, you must ensure that the structure referenced by ioMessageP remains in memory until the asynchronous call completes.

**Comments**  The message data is stored into the data field of the TelSmsDeliveryMessageType structure referenced by ioMessageP. If the complete message data is too large to fit into the allocated size of the data field, the message data is truncated and this function returns the telErrBufferSize error. The dataSize field of the structure is always updated to contain the actual size, in bytes, of the message data.

The originating address string is stored into the originatingData field of the TelSmsDeliveryMessageType structure referenced by ioMessageP. If the complete string is too large to fit into the allocated size of the originatingData field,
the string is truncated (and ends with the null terminator character) and this function returns the telErrBufferSize error. The originatingAddressSize field of the structure is always updated to contain the actual size of the string.

Before calling this function, you need to allocate a number of fields and structures in and related to the TelSmsDeliveryMessage structure:

- Allocate each address field with a size of at least kTelMaxPhoneNumberLen + 1.
  For example, for a GSM message, you must allocate the originatingAddress and serviceCenterNumber fields in the TelSmsDeliveryAdvancedGSMTYPE structure in the TelSmsDeliveryMessage structure.

- Allocate the message field data to have the maximum size of a message on the current network. You can determine this value by calling the TelSmsGetDataMaxSize function.

- Allocate at least one TelSmsExtensionType structure in the TelSmsDeliveryMessage structure. You must have at least one extension structure, even if your message has only a single part. If you do not allocate enough extensions for the message, TelSmsReadMessage returns an error. Palm recommends allocating between 3 and 5 extensions for a message.

- You should not allocate a pointer for user extension data. If you receive user extension data, the user extension pointer will reference a block in the message data. Do not deallocate the user extension data when you release the structure.

Before using this function, you should verify that it is available by calling the TelIsSmsServiceAvailable macro.

**Compatibility**
 Implemented only if 4.0 New Feature Set is present.

**See Also**
 TelSmsSelectStorage
**TelSmsReadMessages**

**Purpose**
Retrieve a range of delivered messages from the currently selected storage area.

**Declared In**
TelephonyMgr.h

**Prototype**
Err TelSmsReadMessages(UInt16 iRefnum,
TelAppID iAppId,
TelSmsReadMessagesType *ioParamP,
UInt16 *ioTransIdP)

**Parameters**
- **-> iRefnum**
The telephony manager library reference number.

- **-> iAppId**
The telephone application attachment identifier for your application.

- **<-> ioParamP**
A pointer to a TelSmsReadMessagesType structure.

  On input, the first field of this structure specifies the index of the first message to retrieve. Message indexes are zero-based.

  On input the count field of this structure specifies the allocated size of the messagesP buffer. Upon return, the count field specifies the actual number of messages that were available, even if that many could not fit into the buffer.

  The messagesP buffer must contain pointers to TelSmsDeliveryMessageType structures that have been allocated. Each of these structures must be initialized as described for the TelSmsReadMessage function.

- **<-> ioTransIdP**
Set the value of this parameter to NULL to cause the function to execute synchronously.
If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**

Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- **returnCode**
  errNone upon success or an error code upon failure.

- **transId**
  The transaction ID of the operation.

- **paramP**
  Points to the `TelSmsReadMessagesType` structure passed to this function in the `ioParamP` parameter.

- **functionId**
  `kTelSmsReadMessagesMessage`

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` and all of the structures referenced by it remain in memory until the asynchronous call completes.

---

**Comments**

If the message data or originating address string data for any of the retrieved messages is larger than the allocated size of its corresponding buffer, the data is truncated into the buffer, and this function returns the `telErrBufferSize` error.

For more information about using this function and allocating structures for its use, see the Comments description for the `TelSmsReadMessage` function.

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable` macro.
### Compatibility
Implemented only if 4.0 New Feature Set is present.

### See Also
TelSmsSelectStorage

## TelSmsReadReport

### Purpose
Read a report from the currently selected storage area.

### Declared In
TelephonyMgr.h

### Prototype
```
Err TelSmsReadReport(UInt16 iRefnum, TelAppID iAppId, TelSmsReportType *ioReportP, UInt16 *ioTransIdP)
```

### Parameters
- **-> iRefnum**
  The telephony manager library reference number.

- **-> iAppId**
  The telephone application attachment identifier for your application.

- **<- ioReportP**
  A pointer to a TelSmsReportType structure.
  
  On input the index field of this structure contains the index of the message that you want retrieved. Message indexes are zero-based.

  On input, the dataSize field specifies the allocated size of the data buffer, and the originatingAddressSize field specifies the allocated size of the originatingAddress string. Upon return, each size field specifies the complete size of the data that was stored into the buffer, even the data had to be truncated to fit.

- **<- ioTransIdP**
  Set the value of this parameter to NULL to cause the function to execute synchronously.

  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
Telephony SMS
Telephony SMS Functions

Synchronous Result
Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result
The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode`: `errNone` upon success or an error code upon failure.
- `transId`: The transaction ID of the operation.
- `paramP`: Points to the `TelSmsReportType` structure passed to this function in the `ioReportP` parameter.
- `functionId`: `kTelSmsReadReportMessage`

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `iEntryP` remains in memory until the asynchronous call completes.

Comments
The report data is stored into the `data` field of the `TelSmsReportType` structure referenced by `ioReportP`. If the complete message data is too large to fit into the allocated size of the `data` field, the report data is truncated and this function returns the `telErrBufferSize` error. The `dataSize` field of the structure is always updated to contain the actual size, in bytes, of the report data.

The originating address string is stored into the `originatingData` field of the `TelSmsReportType` structure referenced by `ioReportP`. If the complete string is too large to fit into the allocated size of the `originatingData` field, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The `originatingAddressSize` field of the structure is always updated to contain the actual size of the string.

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable` macro.
Telephony SMS

Telephony SMS Functions

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

TelSmsSelectStorage, TelSmsSendMessage

TelSmsReadReports

Purpose

Retrieve a range of reports from the currently selected storage.

Declared In

TelephonyMgr.h

Prototype

Err TelSmsReadReports(UInt16 iRefnum, TelAppID iAppId, TelSmsReadReportsType *ioParamP, UInt16 *ioTransIdP)

Parameters

-> iRefnum
   The telephony manager library reference number.

-> iAppId
   The telephone application attachment identifier for your application.

<-> ioParamP
   A pointer to a TelSmsReadReportsType structure.
   On input, the first field of this structure specifies the index of the first report to retrieve. Report indexes are zero-based.
   On input the count field of this structure specifies the allocated size of the reportsP buffer. Upon return, the count field specifies the actual number of reports that were available, even if that many could not fit into the buffer.
   The reportsP buffer must contain pointers to TelSmsReportType structures that have been allocated. Each of these structures must be initialized as described for the TelSmsReadReport function.

<-> ioTransIdP
   Set the value of this parameter to NULL to cause the function to execute synchronously.
If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**

The following fields are updated in the `TelEventTypet` event that is sent when the operation completes:

- **returnCode**
  - errNone upon success or an error code upon failure.
- **transId**
  - The transaction ID of the operation.
- **paramP**
  - Points to the `TelSmsReadReportsType` structure passed to this function in the `ioParamP` parameter.
- **functionId**
  - `kTelSmsReadReportsMessage`

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` and all of the structured referenced by it remain in memory until the asynchronous call completes.

**Comments**
If the report data or originating address string data for any of the retrieved messages is larger than the allocated size of its corresponding buffer, the data is truncated into the buffer, and this function returns the `telErrBufferSize` error. For more information, see the Comments description for the `TelSmsReadReport` function.

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable` macro.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
`TelSmsSelectStorage`, `TelSmsSendMessage`
TelSmsReadSubmittedMessage

Purpose
Read a previously submitted message that was kept on the phone after being sent.

Declared In
TelephonyMgr.h

Prototype
Err TelSmsReadSubmittedMessage(UInt16 iRefnum,
TelAppID iAppId,
TelSmsSubmittedMessageType *ioMessageP,
UInt16 *ioTransIdP)

Parameters
-> iRefnum The telephony manager library reference number.
-> iAppId The telephone application attachment identifier for your application.
<-> ioMessageP A pointer to a TelSmsSubmitMessageType structure that is filled in with the message.

On input, the index field contains the index of the message that you want retrieved. Message indexes are zero-based.

Upon return, the message field structure is filled in with the message information. You must initialize the buffer size fields of this structure prior to calling this function, including the dataSize, callbackNumberSize, serviceCenterNumberSize, and destinationAddressSize fields. Each of these fields is initialized with the allocated size of its corresponding buffer.

<-> ioTransIdP Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
Telephony SMS

Telephony SMS Functions

Synchronous Result

Returns errNone if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the TelEventType event that is sent when the operation completes:

- **returnCode**: errNone upon success or an error code upon failure.
- **transId**: The transaction ID of the operation.
- **paramP**: Points to the TelSmsSubmitMessageType structure passed to this function in the ioMessageP parameter.
- **functionId**: kTelSmsSubmittedMessageMessage

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by iEntryP remains in memory until the asynchronous call completes.

Comments

The message data is stored into the data field of the TelSmsSubmitMessageType structure referenced by ioMessageP. If the complete message data is too large to fit into the allocated size of the data field, the end of the message data is truncated and this function returns the telErrBufferSize error. The dataSize field of the structure is always updated to contain the actual size, in bytes, of the message data.

The same strategy applies to the callbackNumber buffer and callbackNumberSize fields, the destinationAddress buffer and destinationAddressSize fields, and the serviceCenterNumber buffer and serviceCenterNumberSize fields. If the size of the data for any of the buffer fields exceeds the allocated length of the buffer, the end of the data is truncated and this function returns the telErrBufferSize error. Each of the size fields is always updated to contain the complete size of the data intended for the buffer.
Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable` macro.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
TelSmsSelectStorage, TelSmsSendMessage

**TelSmsReadSubmittedMessages**

**Purpose**
Retrieve a range of submitted messages from the currently selected storage area.

**Declared In**
TelephonyMgr.h

**Prototype**
```c
Err TelSmsReadSubmittedMessages(UInt16 iRefnum, TelAppID iAppId, TelSmsReadSubmittedMessagesType *ioParamP, UInt16 *ioTransIdP)
```

**Parameters**
- `iRefnum` The telephony manager library reference number.
- `iAppId` The telephone application attachment identifier for your application.
- `ioParamP` A pointer to a `TelSmsReadMessagesType` structure.

On input, the first field of this structure specifies the index of the first message to retrieve. Message indexes are zero-based.

On input, the count field of this structure specifies the allocated size of thesubmittedsP buffer. Upon return, the count field specifies the actual number of messages that were available, even if that many could not fit into the buffer.
The submittedsP buffer must contain pointers to **TelSmsSubmittedMessageType** structures that have been allocated. Each of these structures must be initialized as described for the **TelSmsReadSubmittedMessage** function.

**<-> ioTransIdP** Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**

Returns **errNone** if the function was successful or returns an error code if not successful.

**Asynchronous Result**

The following fields are updated in the **TelEventType** event that is sent when the operation completes:

- **returnCode**  
  errNone upon success or an error code upon failure.

- **transId**  
  The transaction ID of the operation.

- **paramP**  
  Points to the **TelSmsReadSubmittedMessagesType** structure passed to this function in the ioParamP parameter.

- **functionId**  
  kTelSmsReadSubmittedMessagesMessage

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by ioParamP and all of the structured referenced by it remain in memory until the asynchronous call completes.

**Comments**

If the message data or any of the other variable-length data for any of the retrieved messages is larger than the allocated size of its corresponding buffer, the end of the data is truncated, and this
function returns the telErrBufferSize error. For more information, see the Comments description for the TelSmsReadSubmittedMessage function.

Before using this function, you should verify that it is available by calling the TelIsSmsServiceAvailable macro.

Compatibility
 Implemented only if 4.0 New Feature Set is present.

See Also
 TelSmsSelectStorage, TelSmsSendMessage

TelSmsSelectStorage

Purpose
 Select a storage area on the phone as the current storage area.

Declared In
 TelephonyMgr.h

Prototype
 Err TelSmsSelectStorage(UInt16 iRefnum, TelAppID iAppId, UInt8 iStorageId, UInt16 *ioTransIdP)

Parameters
 -> iRefnum  The telephony manager library reference number.

-> iAppId  The telephone application attachment identifier for your application.

-> ioStorageId  The ID of the storage area. This must be one of the SMS Storage ID Constants.

<-> ioTransIdP  Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result
 Returns errNone if the function was successful or returns an error code if not successful.
Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- **returnCode**: `errNone` upon success or an error code upon failure.
- **transId**: The transaction ID of the operation.
- **paramP**: Points to the unsigned integer value passed to this function in the `ioStorageId` parameter.
- **functionId**: `kTelSmsSelectStorageMessage`

Comments

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable` macro.

Compatibility

Implemented only if [4.0 New Feature Set](#) is present.

See Also

`TelSmsGetAvailableStorage`, `TelSmsGetSelectedStorage`

TelSmsSendManualAcknowledge

Purpose

Send a manual acknowledgment of a previously received message. Note that this function is not supported on GSM networks.

Declared In

`TelephonyMgr.h`

Prototype

```c
Err TelSmsSendManualAcknowledge(UInt16 iRefnum, TelAppID iAppId, TelSmsManualAckType *ioAckP, UInt16 *ioTransIdP)
```

Parameters

- `iRefnum`: The telephony manager library reference number.
- `iAppId`: The telephone application attachment identifier for your application.
- `ioAckP`: A pointer to a structure of type `TelSmsManualAckType`. The fields of this structure specify information about the message being acknowledged.
Upon return, the `messageId` field is filled in with the ID of the acknowledgment.

`<-> ioTransIdP` Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**
Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**
The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode` errNone upon success or an error code upon failure.
- `transId` The transaction ID of the operation.
- `paramP` Points to the `TelSmsManualAckType` structure passed to this function in the `ioAckP` parameter.
- `functionId` `kTelSmsSendManualAcknowledgeMessage`

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `iEntryP` remains in memory until the asynchronous call completes.

**Comments**
Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable` macro.

**Compatibility**
Implemented only if `4.0 New Feature Set` is present.
TelSmsSendMessage

Purpose
Send an SMS message.

Declared In
TelephonyMgr.h

Prototype
Err TelSmsSendMessage(UInt16 iRefnum, TelAppID iAppId, TelSmsSendMessageType *ioMessageP, UInt16 *ioTransIdP)

Parameters
- iRefnum
  The telephony manager library reference number.
- iAppId
  The telephone application attachment identifier for your application.
- ioMessageP
  A pointer to a structure of type TelSmsSendMessageType.
  On input, the message field of this structure contains a TelSmsSubmitMessageType with the message to send. You must also allocate and zero at least one TelSmsExtensionType structure for the multi-part information.
  On output, the messageId field of this structure is filled in with the ID that was assigned to the sent message.
- ioTransIdP
  Set the value of this parameter to NULL to cause the function to execute synchronously.
  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result
Returns errNone if the function was successful or returns an error code if not successful.

Asynchronous Result
The following fields are updated in the TelEventType event that is sent when the operation completes:
**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioMessageP` remains in memory until the asynchronous call completes.

**Comments**

You need to make multiple calls to the `TelSmsSendMessage` function to send your message:

- The first call to `TelSmsSendMessage` does not actually send the message. It computes the number of parts and fills in the multi-part extension structures in the `TelSmsSendMessageType` structure. Note that you must allocate at least one extension structure, even for single-part messages.

- Subsequent calls to `TelSmsSendMessage` actually send the data.

- To send an entire message, you need to call `TelSmsSendMessage` in a loop. Terminate the loop when an error occurs, or when the `byteSend` field of the first `TelSmsExtensionType` structure has the same value as the `dataSize` field of the `TelSmsSendMessageType` structure that represents the message. For example:

```c
while (!TelSmsSendMessage(...) &&
       msg.extensionP[0].extension.mp.byteSend != dataSize);
```

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable` macro.
Compatibility  Implemented only if 4.0 New Feature Set is present.

See Also  TelSmsSelectStorage, TelCfgSetSmsCenter
Telephony Phone Book

This chapter describes the phone book service set of the telephony API.

For more information about the telephony manager basic services and the different service sets, see Chapter 68, “Telephony Basic Services.”

This chapter describes:

• Telephony Phone Book Data Structures
• Telephony Phone Book Constants
• Telephony Phone Book Functions

For more information about using the telephony manager, see Chapter 10, “Telephony Manager,” in Palm OS Programmer’s Companion, vol. II, Communications.

Telephony Phone Book Data Structures

This chapter describes the data structures used with the phone book service set of the telephony API.

TelPhbEntryType

The TelPhbEntryType structure describes a single entry in a phone book.

typedef struct _TelPhbEntryType
{
    UInt16   phoneIndex;
    Char     *fullName;
    UInt8    fullNameSize;
    Char     *dialNumber;
}
Field Descriptions

- phoneIndex
  The index of the entry in the phone’s phone book. This index is always zero-based.
  The telephony manager is responsible for converting this index into the physical index in the phone, if required.

<- fullName
  A buffer into which the retrieved full name of the entry is stored.
  This string is stored using the local character set of the Palm™ handheld device. The telephony manager is responsible for converting the character set, if required.
  Note that if this buffer is too small to contain the entire retrieved string, the string is truncated and the function using this structure generates a `telErrBufferSize` error.

<-> fullNameSize
  The size of the `fullName` string.
  When the structure is used as an input parameter, this is the allocated size, in bytes, of the `fullName` buffer.
  Upon return, this is the actual size of the string, including the null terminator character. If the `fullName` buffer is too small to contain all of the retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.
TelPhbGetAvailablePhonebooks

The TelPhbGetAvailablePhonebooks functions uses the TelPhbGetAvailablePhonebooksType structure to return a list of the phone books available on the phone.

```c
typedef struct _TelPhbGetAvailablePhonebooksType
{
    Uint16    count;
    Uint8     *phonebooksP;
} TelPhbGetAvailablePhonebooksType
```

<- dialNumber A buffer into which the retrieved telephone number of the entry is stored.

Note that if this buffer is too small to contain the entire retrieved string, the string is truncated, and the function using this structure generates a telErrBufferSize error.

<-> dialNumberSize The size of the dialNumber string.

When the structure is used as an input parameter, this is the allocated size, in bytes, of the dialNumber buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the dialNumber buffer is too small to contain all of the retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a telErrBufferSize error.
Field Descriptions

<-> count

The number of entries in the array referenced by phonebooksP.

When the structure is used as an input parameter, this is the allocated size, in bytes, of the phonebooksP buffer. Upon return, this is the actual size of the retrieved data.

Upon return, this is the actual number of phone book IDs that could be retrieved. If the phonebooksP buffer is too small to contain all of the IDs, this field is assigned the actual count, and the function using this structure generates a telErrBufferSize error.

<-> phonebooksP

An array of retrieved phone book IDs. Each ID is one of the Phone Book Type Constants.

TelPhbGetEntriesType

The TelPhbGetEntries function uses the TelPhbGetEntriesType structure to return a list of phone entries.

typedef struct _TelPhbGetEntriesType
{
    UInt16           first;
    UInt16           count;
    TelPhbEntryType  *entriesP;
} TelPhbGetEntriesType

Field Descriptions

-> first

The index of the first entry in the array referenced by entriesP.
TelPhbGetEntryCountType

The TelPhbGetEntryCount function uses the TelPhbGetEntryCountType structure to return information about the entries in the currently selected phone book.

typedef struct _TelPhbGetEntryCountType
{
    UInt16     slots;
    UInt16     count;
} TelPhbGetEntryCountType

Field Descriptions

<- slots The total number of entry slots available in the phone book.

c count The number of filled slots in the phone book.

TelPhbGetEntryMaxSizesType

The TelPhbGetEntryMaxSizes function uses the TelPhbGetEntryMaxSizesType structure to return size information about the entries in the currently selected phone book.

typedef struct _TelPhbGetEntryMaxSizeType
{
    UInt8          fullNameMaxSize;
    UInt8          dialNumberMaxSize;
} TelPhbGetEntryMaxSizesType
Field Descriptions

< - fullNameMaxSize  The largest size of any fullName field in the phone book.
< - dialNumberMaxSize  The largest size of any dialNumber field in the phone book.

Telephony Phone Book Constants

This section describes the constants used with the phone book service set of the telephony API.

Phone Book Type Constants

The phone book type constants specify the type of phone book that is currently selected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTelPhbFixedPhonebook</td>
<td>0</td>
<td>The phone book stored on the phone.</td>
</tr>
<tr>
<td>kTelPhbSimPhonebook</td>
<td>1</td>
<td>The phone book stored on the SIM card.</td>
</tr>
<tr>
<td>kTelPhbPhonePhonebook</td>
<td>2</td>
<td>The phone book stored on the phone.</td>
</tr>
<tr>
<td>kTelPhbLastDialedPhonebook</td>
<td>3</td>
<td>The phone book from which a telephone number was most recently dialed.</td>
</tr>
<tr>
<td>kTelPhbSimAndPhonePhonebook</td>
<td>4</td>
<td>The combined phone and SIM card phone books.</td>
</tr>
</tbody>
</table>
Telephony Phone Book Functions

This section describes the functions used with the phone book service set of the telephony API.

**TelPhbAddEntry**

**Purpose**
Add or replace an entry in the currently selected phone book.

**Declared In**
TelephonyMgr.h

**Prototype**
Err TelPhbAddEntry(UInt16 iRefnum, 
TelAppID iAppId, TelPhbEntryType *iEntryP, 
UInt16 *ioTransIdP)

**Parameters**
- **iRefnum**
The telephony manager library reference number.
- **iAppId**
The telephone application attachment identifier for your application.
- **iEntryP**
A pointer to a TelPhbEntryType structure that contains the new entry information.
- **ioTransIdP**
Set the value of this parameter to NULL to cause the function to execute synchronously.

### Constant Value Description

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kTelPhbAdaptorPhonebook</td>
<td>5</td>
<td>The phone book stored on the telephone adaptor.</td>
</tr>
<tr>
<td>kTelPhbFirstOemPhonebook</td>
<td>6</td>
<td>The ID of the first OEM phone book. You can specify additional OEM phone books by incrementing this value; for example, the second OEM phone book is specified as: kTelPhbFirstOemPhonebook +1</td>
</tr>
</tbody>
</table>
If this parameter is not **NULL**, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

### Synchronous Result

Returns **errNone** if the function was successful or returns an error code if not successful.

### Asynchronous Result

The following fields are updated in the **TelEventType** event that is sent when the operation completes:

- **returnCode**
  - **errNone** upon success or an error code upon failure.
- **transId**
  - The transaction ID of the operation.
- **paramP**
  - Points to the **TelPhbEntryType** structure passed to this function in the **iEntryP** parameter.
- **functionId**
  - **kTelPhbAddEntryMessage**

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by **iEntryP** remains in memory until the asynchronous call completes.

### Comments

The **phoneIndex** field of the **TelPhbEntryType** structure referenced by **iEntryP** specifies the index at which to write the entry.

Before using this function, you should verify that it is available by calling the **TelIsPhbServiceAvailable** macro.

### Compatibility

Implemented only if **4.0 New Feature Set** is present.

### See Also

**TelPhbDeleteEntry**, **TelPhbSelectPhonebook**
**TelPhbDeleteEntry**

**Purpose** Deletes an entry from the currently selected phone book.

**Declared In** TelephonyMgr.h

**Prototype**

```c
Err TelPhbDeleteEntry(UInt16 iRefnum,
TelAppID iAppId, UInt16 iEntryIndex,
UInt16 *ioTransIdP)
```

**Parameters**

- `iRefnum` The telephony manager library reference number.
- `iAppId` The telephone application attachment identifier for your application.
- `iEntryIndex` The zero-based, logical index of the entry that you want deleted. The Telephony Manager computes the physical index.
- `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode` `errNone` upon success or an error code upon failure.
- `transId` The transaction ID of the operation.
- `paramP` Points to the unsigned integer value passed to this function in the `iEntryIndex` parameter.
- `functionId` `kTelPhbDeleteEntryMessage`
WARNING! When using this function asynchronously, you must ensure that the structure referenced by `iEntryP` remains in memory until the asynchronous call completes.

Comments Before using this function, you should verify that it is available by calling the `TelIsPhbServiceAvailable` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelPhbAddEntry`, `TelPhbSelectPhonebook`

**TelPhbGetAvailablePhonebooks**

Purpose Retrieve the list of all phone books available on the phone.

Declared In `TelephonyMgr.h`

Prototype `Err TelPhbGetAvailablePhonebooks(UInt16 iRefnum, TelAppID iAppId, TelPhbGetAvailablePhonebooksType *ioParamP, UInt16 *ioTransIdP)`

Parameters

- `iRefnum` The telephony manager library reference number.
- `iAppId` The telephone application attachment identifier for your application.
- `ioParamP` A pointer to a `TelPhbGetAvailablePhonebooksType` structure that lists the available phone books. On input, the `count` field of this structure specifies the allocated size of the `phonebookP` buffer. Upon return, the `count` field specifies the actual number of entries retrieved, even if they were truncated to fit into the buffer.
<-> ioTransIdP Set the value of this parameter to NULL to cause the function to execute synchronously.

   If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**

Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**

The following fields are updated in the TelEventType event that is sent when the operation completes:

- **returnCode** errNone upon success or an error code upon failure.
- **transId** The transaction ID of the operation.
- **paramP** Points to the TelPhbGetAvailablePhonebooksType structure passed to this function in the iEntryP parameter.
- **functionId** kTelPhbGetAvailablePhonebooksMessage

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by ioParamP remains in memory until the asynchronous call completes.

**Comments**

The phone book IDs are stored into the phonebookP field of the TelPhbGetAvailablePhonebooksType structure referenced by ioParamP. If the phonebookP buffer is too small to contain all of the IDs, the information is truncated and this function returns the telErrBufferSize error. The count field of the structure is always updated to contain the actual number of entries that were retrieved.

Before using this function, you should verify that it is available by calling the TelIsPhbServiceAvailable macro.
### TelPhbGetEntries

**Purpose**
Retrieve a range of entries from the currently selected phone book.

**Declared In**
TelephonyMgr.h

**Prototype**
Err TelPhbGetEntries(UInt16 iRefnum, TelAppID iAppId, TelPhbGetEntriesType *ioParamP, UInt16 *ioTransIdP)

**Parameters**
- **iRefnum**
The telephony manager library reference number.
- **iAppId**
The telephone application attachment identifier for your application.
- **ioParamP**
A pointer to a `TelPhbGetEntriesType` structure that is updated with the phone book entry information. The first entry retrieved is specified in the `first` field of this structure, which is zero-based; the number of entries retrieved is specified by the `count` field. Thus, the last entry retrieved is specified by:

  \[
  \text{ioParamP}\rightarrow\text{first} + \text{ioParamP}\rightarrow\text{count} - 1
  \]

  Upon return, the `count` field of the structure is the number of entries that were actually retrieved.

  The `entriesPtr` field of this structure is a buffer that you allocate to contain the required number of pointers. Each pointer references a `TelPhbEntryType` structure that you must also preallocate.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
TelPhbSelectPhonebook
On input, the fullNameSize and dialNumberSize fields of this structure specify the allocated sizes of the fullName and dialNumber buffers. Upon return, the fullNameSize and dialNumberSize fields specify the actual sizes of the buffers, even if a string was truncated to fit into the buffer.

$\text{ioTransIdP}$ Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**

Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**

The following fields are updated in the *TelEvent* event that is sent when the operation completes:

- `returnCode` errNone upon success or an error code upon failure
- `transId` The transaction ID of the operation.
- `paramP` Points to the *TelPhbGetEntriesType* structure passed to this function in the `ioEntriesP` parameter.
- `functionId` `kTelPhbGetEntriesMessage`

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` and any structures that it references remain in memory until the asynchronous call completes.

**Comments**

The phone book information is stored into the *TelPhbEntryType* structures that you preallocate and refer to in the `entriesP` field of
the `TelPhbGetEntriesType` referenced by the `ioParamP` parameter.

If any buffer in any of the `TelPhbEntryType` structures is too small, the string intended for that buffer is truncated, and this function returns the `telErrBufferSize` error. In any case, the `fullNameSize` and `dialNumberSize` fields of each `TelPhbEntryType` structure contain the actual size of their respective strings.

If any entries in the specified range are empty, the entry is not retrieved, and the `count` value in the structure is updated.

Before using this function, you should verify that it is available by calling the `TelIsPhbServiceAvailable` macro.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
`TelPhbGetEntry`, `TelPhbSelectPhonebook`

### `TelPhbGetEntry`

**Purpose**
Retrieve one entry from the currently selected phone book.

**Declared In**
`TelephonyMgr.h`

**Prototype**

```c
Err TelPhbGetEntry(UInt16 iRefnum,
                   TelAppID iAppId, TelPhbEntryType *ioEntryP,
                   Uint16 *ioTransIdP)
```

**Parameters**

- `-> iRefnum` The telephony manager library reference number.
- `-> iAppId` The telephone application attachment identifier for your application.
- `<-> ioEntryP` A pointer to a `TelPhbEntryType` structure that is updated with the phone book entry information.
On input, the fullNameSize and dialNumberSize fields of this structure specify the allocated sizes of the fullName and dialNumber buffers. Upon return, the fullNameSize and dialNumberSize fields specify the actual sizes of the buffers, even if a string was truncated to fit into the buffer.

```
<-> ioTransIdP
```

Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**

Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result**

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode` errNone upon success or an error code upon failure.
- `transId` The transaction ID of the operation.
- `paramP` Points to the `TelPhbEntryType` structure passed to this function in the `ioEntryP` parameter.
- `functionId` `kTelPhbGetEntryMessage`

**Comments**

The phone book information is stored into the `TelPhbEntryType` that you preallocate. If either buffer in the structure is too small, the string intended for that buffer is truncated, and this function returns the `telErrBufferSize` error. In any case, the fullNameSize and dialNumberSize fields of the structure contain the actual size of their respective strings.

Before using this function, you should verify that it is available by calling the `TelIsPhbServiceAvailable` macro.
### Compatibility
Implemented only if [4.0 New Feature Set](#) is present.

### See Also
[TelPhbGetEntries](#), [TelPhbSelectPhonebook](#)

## TelPhbGetEntryCount

### Purpose
Retrieve the total number of entries, and the number of filled entries in the currently selected phone book.

### Declared In
TelephonyMgr.h

### Prototype
```
Err TelPhbGetEntryCount(UInt16 iRefnum,
TelAppID iAppId,
TelPhbGetEntryCountType *oParamP,
UInt16 *ioTransIdP)
```

### Parameters
- **-> iRefnum**
  The telephony manager library reference number.
- **-> iAppId**
  The telephone application attachment identifier for your application.
- **<- oParamP**
  A pointer to a `TelPhbGetEntryCountType` structure that is updated with information about the number of entries in the phone book.
- **<- ioTransIdP**
  Set the value of this parameter to NULL to cause the function to execute synchronously.
  
  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

### Synchronous Result
Returns `errNone` if the function was successful or returns an error code if not successful.

### Asynchronous Result
The following fields are updated in the `TelEventType` event that is sent when the operation completes:
WARNING! When using this function asynchronously, you must ensure that the structure referenced by oParamP remains in memory until the asynchronous call completes.

Comments
The total number of slots and the number of filled slots in the currently selected phone book are stored in the TelPhbGetEntryCountType structure referenced by oParamP. Before using this function, you should verify that it is available by calling the TelIsPhbServiceAvailable macro.

Compatibility
Implemented only if 4.0 New Feature Set is present.

See Also
TelPhbSelectPhonebook
TelPhbGetEntryMaxSizes

Purpose
Retrieves the maximum buffer sizes of any entries in the currently selected phone book.

Declared In
TelephonyMgr.h

Prototype
Err TelPhbGetEntryMaxSizes(UInt16 iRefnum,
TelAppID iAppId,
TelPhbGetEntryMaxSizesType *oParamP,
UInt16 *ioTransIdP)

Parameters
- iRefnum The telephony manager library reference number.
- iAppId The telephone application attachment identifier for your application.
- oParamP A pointer to a TelPhbGetEntryMaxSizesType structure that is updated with information about the maximum buffer sizes of entries in the phone book.
- ioTransIdP Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result
Returns errNone if the function was successful or returns an error code if not successful.

Asynchronous Result
The following fields are updated in the TelEventType event that is sent when the operation completes:

- returnCode errNone upon success or an error code upon failure.
- transId The transaction ID of the operation.
paramP  Points to the `TelPhbGetEntryMaxSizesType` structure passed to this function in the oParamP parameter.

functionId  kTelPhbGetEntryMaxSizesMessage

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by oParamP remains in memory until the asynchronous call completes.

**Comments**
The maximum size of any full name entry and the maximum size of any telephone number entry in the currently selected phone book are stored in the `TelPhbGetEntryMaxSizesType` structure referenced by oParamP.

Before using this function, you should verify that it is available by calling the `TelIsPhbServiceAvailable` macro.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
TelPhbGetEntries, TelPhbGetEntry

**TelPhbGetSelectedPhonebook**

**Purpose**
Retrieve the ID of the currently selected phone book.

**Declared In**
TelephonyMgr.h

**Prototype**

```c
Err TelPhbGetSelectedPhonebook(UInt16 iRefnum, TelAppID iAppId, UInt8 *oPhbIdP, UInt16 *ioTransIdP)
```

**Parameters**
- `-> iRefnum`  The telephony manager library reference number.
- `-> iAppId`  The telephone application attachment identifier for your application.
Telephony Phone Book

Telephony Phone Book Functions

<- oPhbIdP    A pointer to an unsigned byte value. Upon return, this is filled in with the identifier of the currently selected phone book. The identifier is one of the Phone Book Type Constants.

<> ioTransIdP  Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns errNone if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the TelEventType event that is sent when the operation completes:

returnCode    errNone upon success or an error code upon failure.

transId       The transaction ID of the operation.

paramP        Points to the unsigned integer value passed to this function in the oPhbIdP parameter.

functionId    kTelPhbGetSelectedPhonebookMessage

WARNING!     When using this function asynchronously, you must ensure that the value referenced by oPhbIdP remains in memory until the asynchronous call completes.

Comments

Before using this function, you should verify that it is available by calling the TelIsPhbServiceAvailable macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

TelPhbSelectPhonebook
**TelPhbSelectPhonebook**

**Purpose**  
Make the specified phone book the currently selected phone book.

**Declared In**  
TelephonyMgr.h

**Prototype**  
```c
Err TelPhbSelectPhonebook(UInt16 iRefnum, 
TelAppID iAppId, UInt8 iPhbId, 
UInt16 *ioTransIdP)
```

**Parameters**

- `- iRefnum`  
The telephony manager library reference number.

- `- iAppId`  
The telephone application attachment identifier for your application.

- `- iPhbId`  
The identifier of the phone book that you want selected as the current phone book. This must be one of the Phone Book Type Constants.

- `ioTransIdP`  
Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**

Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

- `returnCode`  
`errNone` upon success or an error code upon failure.

- `transId`  
The transaction ID of the operation.

- `paramP`  
Points to the unsigned integer value passed to this function in the `iPhbId` parameter.

- `functionId`  
*kTelPhbSelectPhonebookMessage*
Comments Before using this function, you should verify that it is available by calling the TelIsPhbServiceAvailable macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also TelPhbGetAvailablePhonebooks, TelPhbGetSelectedPhonebook
Part IV: Libraries
This chapter provides reference material for the Internet library API:

- Internet Library Data Structures
- Internet Library Constants
- Internet Library Functions

The header file INetMgr.h declares the Internet library API. For more information on the Internet library, see the chapter “Network Communication” in the Palm OS Programmer’s Companion, vol. II, Communications.

**NOTE:** The information in this chapter applies only to version 3.2 or later of the Palm OS® on Palm VII® devices. These features are implemented only if the Wireless Internet Feature Set is present.

**WARNING!** In future OS versions, PalmSource, Inc. does not intend to support or provide backward compatibility for the Internet library API documented in this chapter.

### Internet Library Data Structures

**I\NetCompressionTypeEnum**

The I\NetCompressionTypeEnum enum indicates the type of compression used for data exchanged via a socket. One of these enumerated types is set as the value of the inetSockSettingCompressionTypeID socket setting (a read-only setting).
typedef enum {
    inetCompressionTypeNone = 0,
    inetCompressionTypeBitPacked,
    inetCompressionTypeLZ77
} INetCompressionTypeEnum;

Value Descriptions

<table>
<thead>
<tr>
<th>Enum Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inetCompressionTypeNone</td>
<td>No compression.</td>
</tr>
<tr>
<td>inetCompressionTypeBitPacked</td>
<td>Custom 5-bit compression scheme. This is typically used for data sent from the Palm Web Clipping Proxy server.</td>
</tr>
<tr>
<td>inetCompressionTypeLZ77</td>
<td>Not used; reserved for future use.</td>
</tr>
</tbody>
</table>

**INetConfigNameType**

The INetConfigNameType structure holds the name of an Internet library network **configuration**. A configuration is a set of specific values for the Internet library settings. The Internet library defines a set of built-in configuration aliases for common network setups. These aliases point to configurations instead of holding the actual values themselves. You can use an alias anywhere in the API you would use a configuration. System-defined configuration aliases are listed in “**Configuration Aliases**” on page 1850.

This structure is used in the functions **INetLibConfigIndexFromName**, **INetLibConfigRename**, and **INetLibConfigSaveAs**.

```c
#define inetConfigNameSize 32;

typedef struct {
    Char name[inetConfigNameSize]; // name of configuration
} INetConfigNameType, *INetConfigNamePtr;
```

**Field Description**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>A configuration name (up to 32 characters).</td>
</tr>
</tbody>
</table>
INetContentTypeEnum

The INetContentTypeEnum enum specifies the type of content to be exchanged via a socket. One of these enumerated types is set as the value of the inetSockSettingContentTypeID socket setting (a read-only setting).

```c
typedef enum {
inetContentTypeTextPlain = 0,
inetContentTypeTextHTML,
inetContentTypeImageGIF,
inetContentTypeImageJPEG,
inetContentTypeApplicationCML,
inetContentTypeImagePalmOS,
inetContentTypeOther
} INetContentTypeEnum;
```

Value Descriptions

<table>
<thead>
<tr>
<th>INetContentType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inetContentTypeTextPlain</td>
<td>Not used</td>
</tr>
<tr>
<td>inetContentTypeTextHTML</td>
<td>Not used</td>
</tr>
<tr>
<td>inetContentTypeImageGIF</td>
<td>Not used</td>
</tr>
<tr>
<td>inetContentTypeImageJPEG</td>
<td>Not used</td>
</tr>
<tr>
<td>inetContentTypeApplicationCML</td>
<td>Compressed HTML content (format used by the Palm Web Clipping Proxy server and PQAs)</td>
</tr>
<tr>
<td>inetContentTypeImagePalmOS</td>
<td>Palm OS® bitmap</td>
</tr>
<tr>
<td>inetContentTypeOther</td>
<td>Some undefined content type</td>
</tr>
</tbody>
</table>

INetHTTPAttrEnum

The INetHTTPAttrEnum enum specifies HTTP request and response attributes that are set by INetLibSockHTTPAttrSet and returned by INetLibSockHTTPAttrGet.
typedef enum {

    // Request only attributes
    // The following are ignored unless going through a CTP proxy
    inetHTTPAttrWhichPart, // (W) UInt32 (0 -> N)
    inetHTTPAttrIncHTTP, // (W) UInt32 (Boolean) only applicable
        // when inetHTTPAttrConvAlgorithm set to ctpConvNone
    inetHTTPAttrCheckMailHi, // (W) UInt32
    inetHTTPAttrCheckMailLo, // (W) UInt32
    inetHTTPAttrReqContentVersion, // (W) UInt32, desired content
        // version. Represented as 2 low bytes. Lowest
    byte is
        // minor version, next higher byte is major
    version.
    // Response only attributes
    // Server response info
    inetHTTPAttrRspSize, // (R) UInt32, entire HTTP Response size
        // including header and data
    inetHTTPAttrResult, // (R) UInt32 (ctpErrXXX when using CTP
    Proxy)
    inetHTTPAttrErrDetail, // (R) UInt32 (server/proxy err code
    when
        // using CTP Proxy)
    inetHTTPAttrReason, // (R) Char[]
    // Returned entity attributes
    inetHTTPAttrContentLength, // (R) UInt32
    inetHTTPAttrContentLengthUncompressed, // (R) UInt32 (in
    bytes)
    inetHTTPAttrContentLengthUntruncated, // (R) UInt32
    inetHTTPAttrContentVersion, // (R) UInt32, actual content
    version.
        // Represented as 2 low bytes. Lowest byte is minor
    // version, next higher byte is major version.
    inetHTTPAttrContentCacheID, // (R) UInt32, cacheID for this
    item
    inetHTTPAttrReqSize // (R) UInt32 size of request sent
} INetHTTPAttrEnum;
### Value Descriptions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inetHTTPAttrWhichPart</td>
<td>An index to the part of the response data desired, if the response data is partitioned into chunks. Write-only.</td>
</tr>
<tr>
<td>inetHTTPAttrIncHTTP</td>
<td>A Boolean that, if set, causes HTTP header data to be included as part of the content when retrieving raw data. Applicable only when inetSettingConvAlgorithm is set to ctpConvNone. Write-only.</td>
</tr>
<tr>
<td>inetHTTPAttrCheckMailHi</td>
<td>High-order byte of ID for checking mail. Write-only.</td>
</tr>
<tr>
<td>inetHTTPAttrCheckMailLo</td>
<td>Low-order byte of ID for checking mail. Write-only.</td>
</tr>
<tr>
<td>inetHTTPAttrReqContentVersion</td>
<td>Desired content version. Represented as 2 low bytes. Lowest byte is minor version, next higher byte is major version. Write-only.</td>
</tr>
<tr>
<td>inetHTTPAttrRspSize</td>
<td>Size of entire HTTP (header and data). Read-only.</td>
</tr>
<tr>
<td>inetHTTPAttrResult</td>
<td>Transport protocol error code. Read-only.</td>
</tr>
<tr>
<td>inetHTTPAttrErrDetail</td>
<td>Server/proxy error code when using the Palm Web Clipping Proxy server. Read-only.</td>
</tr>
<tr>
<td>inetHTTPAttrReason</td>
<td>Transport protocol error message. Read-only.</td>
</tr>
<tr>
<td>inetHTTPAttrContentLength</td>
<td>Size of response data. Read-only.</td>
</tr>
<tr>
<td>inetHTTPAttrContentLengthUncompressed</td>
<td>Size of uncompressed response data. Read-only.</td>
</tr>
<tr>
<td>inetHTTPAttrContentLengthUntruncated</td>
<td>Total size of response data (it may have been truncated to less than this). Read-only.</td>
</tr>
</tbody>
</table>
**INetSchemeEnum**

The INetSchemeEnum enum specifies a protocol (http, https, etc.) used by a socket. Specify one of these enumerated types for the INetSockSettingScheme socket setting and for the scheme parameter to the INetLibSockOpen call.

```c
typedef enum {
    inetSchemeUnknown = -1,
    inetSchemeDefault = 0,   // not used
    inetSchemeHTTP, // http:
    inetSchemeHTTPS, // https:
    inetSchemeFTP, // ftp:
    inetSchemeGopher, // gopher:
    inetSchemeFile, // file:
    inetSchemeNews, // news:
    inetSchemeMailTo, // mailto:
    inetSchemePalm, // palm:
    inetSchemePalmCall, // palmcall:
    inetSchemeMail, // not applicable to URLs, but used
        // for the INetLibSockOpen call when
        // creating a socket for mail IO
    inetSchemeMac, // mac: - Mac file system HTML
    inetSchemeFirst = inetSchemeHTTP, // first one
    inetSchemeLast = inetSchemeMail // last one
} INetSchemeEnum;
```

**Value Descriptions**

<table>
<thead>
<tr>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inetSchemeHTTP</td>
<td>Use the HTTP protocol.</td>
</tr>
<tr>
<td>inetSchemeHTTPS</td>
<td>Use the HTTPS protocol (for a secure connection).</td>
</tr>
</tbody>
</table>
INetSettingEnum

The INetSettingEnum enum specifies a setting to be returned or set by the INetLibSettingGet or INetLibSettingSet calls.

typedef enum {
  inetSettingCacheSize, // (RW) UInt32, max size of cache
  inetSettingCacheRef, // (R) DmOpenRef, ref of cache DB
  inetSettingNetLibConfig, // (RW) UInt32, NetLib config to use
  inetSettingRadioID, // (R) UInt32[2], the 64-bit radio ID
  inetSettingBaseStationID, // (R) UInt32, the radio base station Id
  inetSettingMaxRspSize, // (W) UInt32 (in bytes)
  inetSettingConvAlgorithm, // (W) UInt32 (CTPConvEnum)
  inetSettingContentWidth, // (W) UInt32 (in pixels)
}
inetSettingContentVersion, // (W) UInt32, content version
(encoder
    // version)
inetSettingNoPersonalInfo, // (RW) UInt32, send no deviceID/
zipcode
inetSettingUserName,
inetSettingLast
} INetSettingEnum;

Value Descriptions

inetSettingCacheSize Maximum size of cache (in bytes).
inetSettingCacheRef DmOpenRef, reference to cache database. Read-only.
inetSettingNetLibConfig The index of the net library network configuration
to use. This value is saved as part of the
preferences for each Internet library configuration.
A value of 0 means to use the current
configuration.
inetSettingRadioID 64-bit radio ID. Read-only. Used for wireless
connections only.
inetSettingBaseStationID Radio base station ID. Read-only. Used for wireless
connections only.
inetSettingMaxRspSize Maximum response size, in bytes. The default is
1024 bytes. Write-only.
inetSettingConvAlgorithm Content conversion desired. Write-only. Possible
values include:
ctpConvCML (use 5-bit compression scheme),
ctpConvCML8Bit (use 5-bit compression scheme,
but in 8-bit form for debugging),
ctpConvCMLLZ77 (use LZ77 compression
scheme),
ctpConvNone (no conversion; data is returned in
native format)
inetSettingContentWidth Width of the display for content. The default
setting is 160 (pixels). Write-only.
INetSockSettingEnum

The INetSockSettingEnum enum specifies a socket setting to be returned or set by the INetLibSockSettingGet or INetLibSockSettingSet calls.

typedef enum {
    inetSockSettingScheme, // (R) UInt32, INetSchemeEnum
    inetSockSettingSockContext, // (RW) UInt32,
    inetSockSettingCompressionType, // (R) Char[]
    inetSockSettingCompressionTypeID, // (R) UInt32
    // (INetCompressionTypeEnum)
    inetSockSettingContentType, // (R) Char[]
    inetSockSettingContentTypeID, // (R) UInt32
    (INetContentTypeEnum)
    inetSockSettingData, // (R) UInt32, pointer to data
    inetSockSettingDataHandle, // (R) UInt32, handle to data
    inetSockSettingDataOffset, // (R) UInt32, offset to data from handle
    inetSockSettingTitle, // (W) Char[]
    inetSockSettingURL, // (R) Char[]
    inetSockSettingIndexURL, // (RW) Char[]
    inetSockSettingFlags, // (RW) UInt16, one or more of
    // inetOpenURLFlagXXX flags
    inetSockSettingReadTimeout, // (RW) UInt32, read timeout in ticks
    inetSockSettingContentVersion, // (R) UInt32, content version number
    inetSockSettingLast
} INetSockSettingEnum;
### Value Descriptions

<table>
<thead>
<tr>
<th>Value Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inetSockSettingScheme</td>
<td>Requested scheme; one of the INetSchemeEnum values. Read-only.</td>
</tr>
<tr>
<td>inetSockSettingSockContext</td>
<td>Not used.</td>
</tr>
<tr>
<td>inetSockSettingCompressionType</td>
<td>Name of requested compression type. Read-only.</td>
</tr>
<tr>
<td>inetSockSettingCompressionTypeID</td>
<td>Requested compression type; one of the INetCompressionTypeEnum values. Read-only.</td>
</tr>
<tr>
<td>inetSockSettingContentType</td>
<td>String containing the MIME type of the content. Used only on received raw data. Read-only.</td>
</tr>
<tr>
<td>inetSockSettingContentTypeID</td>
<td>Content type of socket data; one of the INetContentTypeEnum values. Read-only.</td>
</tr>
<tr>
<td>inetSockSettingData</td>
<td>Pointer to socket data. Read-only.</td>
</tr>
<tr>
<td>inetSockSettingDataHandle</td>
<td>Handle to socket data. Read-only.</td>
</tr>
<tr>
<td>inetSockSettingDataOffset</td>
<td>Offset to socket data from handle. Read-only.</td>
</tr>
<tr>
<td>inetSockSettingTitle</td>
<td>Web page title. This value is written to the cache (and the Web Clipping Application Viewer uses it later in a history list of cache entries). Write-only.</td>
</tr>
<tr>
<td>inetSockSettingURL</td>
<td>URL of requested data. Read-only.</td>
</tr>
<tr>
<td>inetSockSettingIndexURL</td>
<td>Index (or master) URL of requested data (for cache indexing). This is the topmost web page in a group of hierarchical pages; it serves to group the pages together and to filter cache list results. The Web Clipping Application Viewer sets this to the URL of the active PQA, for all pages linked from the PQA.</td>
</tr>
</tbody>
</table>
**INetStatusEnum**

The INetStatusEnum enum specifies the status of the socket. The status is returned in the `inetSockStatusChangeEvent` event structure and by the call `INetLibSockStatus`.

```c
typedef enum {
    inetStatusNew, // just opened
    inetStatusResolvingName, // looking up host address
    inetStatusNameResolved, // found host address
    inetStatusConnecting, // connecting to host
    inetStatusConnected, // connected to host
    inetStatusSendingRequest, // sending request
    inetStatusWaitingForResponse, // waiting for response
    inetStatusReceivingResponse, // receiving response
    inetStatusResponseReceived, // response received
    inetStatusClosingConnection, // closing connection
    inetStatusClosed, // closed
    inetStatusAcquiringNetwork, // network temporarily
    // unreachable; socket on
    hold
    inetStatusPrvInvalid = 30 // internal value, not returned by
    // INetMgr. Should be last.
} INetStatusEnum;
```
### Internet Library Constants

#### Value Descriptions

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inetStatusNew</td>
<td>Just opened</td>
</tr>
<tr>
<td>inetStatusResolvingName</td>
<td>Looking up host address</td>
</tr>
<tr>
<td>inetStatusNameResolved</td>
<td>Found host address</td>
</tr>
<tr>
<td>inetStatusConnecting</td>
<td>Connecting to host</td>
</tr>
<tr>
<td>inetStatusConnected</td>
<td>Connected to host</td>
</tr>
<tr>
<td>inetStatusSendingRequest</td>
<td>Sending request</td>
</tr>
<tr>
<td>inetStatusWaitingForResponse</td>
<td>Waiting for response</td>
</tr>
<tr>
<td>inetStatusReceivingResponse</td>
<td>Receiving response</td>
</tr>
<tr>
<td>inetStatusResponseReceived</td>
<td>Response received</td>
</tr>
<tr>
<td>inetStatusClosingConnection</td>
<td>Closing connection</td>
</tr>
<tr>
<td>inetStatusClosed</td>
<td>Connection closed</td>
</tr>
<tr>
<td>inetStatusAcquiringNetwork</td>
<td>Network temporarily unreachable; socket on hold</td>
</tr>
<tr>
<td>inetStatusPrvInvalid</td>
<td>Not used</td>
</tr>
</tbody>
</table>

#### Internet Library Constants

### Configuration Aliases

The constants listed here specify Internet library network configuration alias names. Most of the Internet library API requires a configuration index rather than a name. Use `INetLibConfigIndexFromName` to obtain the alias’s index from the name. For more information, see `INetConfigNameType`.

The following aliases are defined for configuration names:
The `inetURLInfoFlag` constants convey information about a URL and are returned by the function `INetLibURLGetInfo`.

<table>
<thead>
<tr>
<th>Alias</th>
<th>Name string</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inetCfgNameDefault</code></td>
<td>&quot;.Default&quot;</td>
<td>Initially points to a generic configuration with no proxy. This uses the configuration set by the user in the Network preferences panel.</td>
</tr>
<tr>
<td><code>inetCfgNameDefWireline</code></td>
<td>&quot;.DefWireline&quot;</td>
<td>Initially points to a generic configuration with no proxy. This uses the configuration set by the user in the Network preferences panel.</td>
</tr>
<tr>
<td><code>inetCfgNameDefWireless</code></td>
<td>&quot;.DefWireless&quot;</td>
<td>Initially points to a generic configuration with no proxy. This uses the configuration set by the user in the Network preferences panel.</td>
</tr>
<tr>
<td><code>inetCfgNameCTPDefault</code></td>
<td>&quot;.CTPDefault&quot;</td>
<td>Initially points to either &quot;.CTPWireless&quot; (on Palm VII® units) or &quot;.CTPWireline&quot; (on all other units). On the Palm VII unit, the Web Clipping Application Viewer application uses this configuration.</td>
</tr>
<tr>
<td><code>inetCfgNameCTPWireline</code></td>
<td>&quot;.CTPWireline&quot;</td>
<td>Initially points to a wireline configuration that uses the Palm Web Clipping Proxy server.</td>
</tr>
<tr>
<td><code>inetCfgNameCTPWireless</code></td>
<td>&quot;.CTPWireless&quot;</td>
<td>Initially points to a wireless configuration that uses the Palm.Net™ wireless system and the Palm Web Clipping Proxy server.</td>
</tr>
</tbody>
</table>
### URL Open Constants

The `inetOpenURLFlag...` constants control how the `INetLibURLOpen` call operates with respect to caching and encryption. These flags are also used for the `inetSockSettingFlags` socket setting.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inetOpenURLFlagLookInCache</code></td>
<td>0x0001</td>
<td>Read data from the cache, if available.</td>
</tr>
<tr>
<td><code>inetOpenURLFlagKeepInCache</code></td>
<td>0x0002</td>
<td>Store the item in the cache, overwriting any other entries with an equivalent URL.</td>
</tr>
<tr>
<td><code>inetOpenURLFlagForceEncOn</code></td>
<td>0x0008</td>
<td>Use encryption even if scheme does not desire it.</td>
</tr>
<tr>
<td><code>inetOpenURLFlagForceEncOff</code></td>
<td>0x0010</td>
<td>Do not use encryption even if scheme desires it.</td>
</tr>
</tbody>
</table>
Internet Library Functions

**INetLibCacheGetObject**

**Purpose**
Returns information about an entry in the cache database, including a handle to the record. Either the URL or the unique ID can be used to find the cache entry.

**Declared In**
INetMgr.h

**Prototype**
Err INetLibCacheGetObject (UInt16 libRefnum, MemHandle clientParamH, UInt8 *urlTextP, UInt32 uniqueID, INetCacheInfoPtr cacheInfoP)

**Parameters**
- **libRefnum**
  Refnum of the Internet library.
- **clientParamH**
  Inet handle allocated by INetLibOpen.
- **urlTextP**
  Pointer to URL text string to find. If this parameter is NULL, then uniqueID is used to find the entry.
- **uniqueID**
  Unique ID of the cache entry to find. This value can be obtained by calling INetLibCacheList. This parameter is ignored if urlTextP is specified.

- **INetCacheInfoPtr**
  Pointer to a structure where information about the cache entry is returned. See the Comments section for details.

**Result**

- **0**
  No error
- **inetErrParamsInvalid**
  One or more of the parameters are invalid.
Comments  The INetCacheInfoPtr type returned from this function is defined as a pointer to an INetCacheInfoType structure, which has the following definition:

```c
typedef struct {
    MemHandle recordH; // handle to the cache record
    INetContentTypeEnum contentType;
    INetCompressionTypeEnum encodingType;
    UInt32 uncompressedDataSize;
    UInt8 flags; // unused
    UInt8 reserved;
    UInt16 dataOffset; // offset to content
    UInt16 dataLength; // size of content
    UInt16 urlOffset; // offset to URL
    UInt32 viewTime; // time last viewed
    UInt32 createTime; // time entry was created
    UInt16 murlOffset; // offset to master URL
} INetCacheInfoType, *INetCacheInfoPtr;
```

Compatibility  Implemented only if Wireless Internet Feature Set is present.

INetLibCacheList

Purpose  Returns an item from the cache list, based on its URL and index in the list.

Declared In  INetMgr.h

Prototype  Err INetLibCacheList (UInt16 libRefnum, MemHandle inetH, UInt8 *cacheIndexURLP, UInt16 *indexP, UInt32 *uidP, INetCacheEntryP cacheP)

Parameters  -> libRefnum       Refnum of the Internet library.
-> inetH           Inet handle allocated by INetLibOpen.
Internet Library
Internet Library Functions

-> cacheIndexURLP
  Pointer to a master URL string. Cache entries indexed with this master URL are returned. The Web Clipping Application Viewer sets the master URL of a cache page to the URL of the active PQA, so all pages linked from the PQA have the same master URL.

<-> indexP
  Pointer to the index of the entry. Specify an index to find entries at this index or higher in the list. Specify NULL to search from the beginning, the first time you call this function. The index of the entry following the one found is returned on exit.

<- uidP
  Pointer to a long value where the unique ID of the found cache entry is returned.

<- cacheP
  Pointer to a structure where information about the found cache entry is returned. See the Comments section for details.

Result

0
  No error

inetErrTypeNotCached
  Cache entry under requested index not found

inetErrParamsInvalid
  The cacheIndexURLP parameter is NULL

inetErrCacheInvalid
  The cache database doesn’t exist

Comments

This function first sorts the list of cache entries by URL. Then it returns in uidP the unique ID of the first cache entry with an index equal to or greater than indexP. The indexP value is updated to point to the next entry upon return.

To generate a complete list of cache entries having the same master URL (as for a history list), call this function repeatedly, always specifying the updated index, until it returns the error inetErrTypeNotCached.
Note that a URL can exist multiple times in the Web Clipping Application Viewer cache database, thus the need for the uidP value.

The INetCacheEntryP type returned from this function is defined as a pointer to an INetCacheEntryType structure, which has the following definition:

```c
typedef struct {
    UInt8 *urlP; // ptr to URL string
    UInt16 urlLen; // length of URL string
    UInt8 *titleP; // ptr to title string
    UInt16 titleLen; // length of title string
    UInt32 lastViewed; // time last viewed
        // seconds since 1/1/1904
    UInt32 firstViewed; // time first viewed
        // seconds since 1/1/1904
} INetCacheEntryType, *INetCacheEntryP;
```

**Compatibility**
Implemented only if Wireless Internet Feature Set is present.

### INetLibCheckAntennaState

**Purpose**
Checks the antenna state and displays a dialog asking the user to raise it if it is down.

**Declared In**
INetMgr.h

**Prototype**
Err INetLibCheckAntennaState(UInt16 refNum)

**Parameters**
- `refNum` Refnum of the Internet library.

**Result**
- 0 The user raised the antenna.
- `netErrUserCancel` The user closed the dialog by tapping Cancel.

This call can also return data manager errors if it fails internally.
Comments
Applications don’t need to check the antenna state by using this call. If an application opens the Internet library, the Internet library checks the antenna state when needed and displays the dialog to prompt the user to raise the antenna.

Compatibility
Implemented only if Wireless Internet Feature Set is present.

**INetLibClose**

### Purpose
Closes up and frees an inet handle. Closes or decrements the open count of the net library.

### Declared In
**INetMgr.h**

### Prototype
```
Err INetLibClose (UInt16 libRefnum, MemHandle inetH)
```

### Parameters
- `libRefnum` Refnum of the Internet library.
- `inetH` Inet handle allocated by INetLibOpen.

### Result

<table>
<thead>
<tr>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error</td>
</tr>
</tbody>
</table>

### Comments
This call must be made by an application when it’s done with the Internet library. It closes any Internet sockets open by the application, disposes the memory referenced by the given inet handle, and calls NetLibClose, if necessary, to close the net Library or decrement its open count.

### Compatibility
Implemented only if Wireless Internet Feature Set is present.

### See Also
INetLibOpen
**INetLibConfigAliasGet**

**Purpose**
Determines to which configuration a built-in alias points.

**Declared In**
INetMgr.h

**Prototype**
Err INetLibConfigAliasGet (UInt16 refNum, 
UInt16 aliasIndex, UInt16 *indexP, 
Boolean *isAnotherAliasP)

**Parameters**
- **libRefnum**: Refnum of the Internet library.
- **aliasIndex**: Index of alias configuration to query. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the array returned by INetLibConfigList, or the index returned by INetLibConfigIndexFromName.
- **indexP**: Pointer where the index of the configuration pointed to by aliasIndex is returned. 0 is returned if aliasIndex does not point to another configuration.
- **isAnotherAliasP**: If *indexP* is the index of another alias configuration, this Boolean is set to true.

**Result**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>inetErrParamsInvalid</td>
<td>aliasIndex is not valid</td>
</tr>
<tr>
<td>inetErrConfigNotAlias</td>
<td>aliasIndex is not an alias configuration</td>
</tr>
</tbody>
</table>

**Compatibility**
Implemented only if **Wireless Internet Feature Set** is present.

**See Also**
INetLibConfigAliasSet
**INetLibConfigAliasSet**

**Purpose** Points any of the built-in aliases (".DefWireline", ".DefWireless", etc.) to a given defined configuration.

**Declared In** INetMgr.h

**Prototype**
```
Err INetLibConfigAliasSet (UInt16 refNum, UInt16 configIndex, UInt16 aliasToIndex)
```

**Parameters**
- `refNum` Refnum of the Internet library.
- `configIndex` Index of configuration to set. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the array returned by **INetLibConfigList**, or the index returned by **INetLibConfigIndexFromName**.
- `aliasToIndex` Index of configuration that the alias identified by `configIndex` is to point to. Specify 0 to remove an existing alias assignment.

**Result**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>inetErrConfigNotAlias</td>
<td><code>configIndex</code> is not an alias configuration</td>
</tr>
<tr>
<td>inetErrParamsInvalid</td>
<td><code>configIndex</code> or <code>aliasToIndex</code> is not valid</td>
</tr>
<tr>
<td>inetErrConfigCantPointToAlias</td>
<td>Alias doesn't point to a real entry</td>
</tr>
</tbody>
</table>

**Compatibility** Implemented only if **Wireless Internet Feature Set** is present.

**See Also** [INetLibConfigAliasGet](#)
## INetLibConfigDelete

### Purpose
Deletes a configuration.

### Declared In
INetMgr.h

### Prototype
Err INetLibConfigDelete (UInt16 refNum, UInt16 index)

### Parameters
- **-> refnum**
  Refnum of the Internet library.
- **-> index**
  Index of configuration to delete. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the array returned by INetLibConfigList, or the index returned by INetLibConfigIndexFromName.

### Result
- **0**
  No error
- **inetErrParamsInvalid**
  Index not valid
- **inetErrConfigCantDelete**
  Attempted to delete an alias configuration

### Compatibility
Implemented only if Wireless Internet Feature Set is present.

### See Also
INetLibConfigIndexFromName, INetLibConfigList
INetLibConfigIndexFromName

Purpose
Returns the index of a configuration given it's name.

Declared In
INetMgr.h

Prototype
Err INetLibConfigIndexFromName (UInt16 refNum, INetConfigNamePtr nameP, UInt16 *indexP)

Parameters
- refnum Refnum of the Internet library.
- nameP Pointer to an INetConfigNameType structure that names the configuration whose index you want to get.
- indexP Pointer where the index of the configuration identified in nameP is returned.

Result
0 No error
inetErrConfigNotFound Could not find requested configuration name

Comments
If you name an alias, this routine returns the index of the alias entry, not the configuration the alias points to. This way the alias can be pointed to a different configuration.

Applications should store the index of the configuration they’re using, rather than the name, so that they won’t be confused if the user edits the name of the configuration from the Preferences panel.

Compatibility
Implemented only if Wireless Internet Feature Set is present.

See Also
INetLibConfigList
**INetLibConfigList**

**Purpose**
Returns an array containing a list of the available Internet library network configurations.

**Declared In**
INetMgr.h

**Prototype**
```c
Err INetLibConfigList (UInt16 refNum,
INetConfigNameType nameArray[],
UInt16 *arrayEntriesP)
```

**Parameters**
- `refnum` Refnum of the Internet library.
- `nameArray` Pointer to an array of `INetConfigNameType` structs that is to be filled in by this routine.
- `arrayEntriesP` On entry, a pointer to the number of entries available in `nameArray`; on exit, a pointer to the total number of entries in the system (which could exceed the size of the array you pass in).

**Result**
- 0 No error

**Comments**
This routine can be used to obtain a list of available configurations for selection by the user.

Note that the built-in alias configurations, which start with a period, should not be displayed to the user as selectable choices. They are designed for internal use by applications that need a predetermined type of service (like ".CTPWireless" for PQA applications.) Also, any configurations that start with an underscore, like "_CTPRAM", should not be displayed. These typically are configurations created by the Internet library for internal use and cannot be edited using the Network preferences panel.

**Compatibility**
Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also**
INetLibConfigMakeActive
INetLibConfigMakeActive

Purpose  Makes the given configuration active without having to close and reopen the Internet library by using INetLibOpen.

Declared In  INetMgr.h

Prototype  Err INetLibConfigMakeActive (UInt16 refNum, MemHandle inetH, UInt16 configIndex)

Parameters  
-> libRefnum  Refnum of the Internet library.
-> inetH  Inet handle allocated by INetLibOpen.
-> configIndex  Index of configuration to activate. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the array returned by INetLibConfigList, or the index returned by INetLibConfigIndexFromName.

Result  
0  No error

Compatibility  Implemented only if Wireless Internet Feature Set is present.

See Also  INetLibConfigSaveAs, INetLibConfigList, INetLibConfigIndexFromName
**INetLibConfigRename**

**Purpose** Renames a configuration.

**Declared In** INetMgr.h

**Prototype**

Err INetLibConfigRename (UInt16 refNum, UInt16 index, INetConfigNamePtr newNameP)

**Parameters**

- `-> libRefnum` Refnum of the Internet library.
- `-> index` Index of configuration to rename. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the array returned by INetLibConfigList, or the index returned by INetLibConfigIndexFromName.
- `-> newNameP` Pointer to an INetConfigNameType structure holding the new name of the configuration. The name cannot start with a period or an underscore.

**Result**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>inetErrConfigBadName</td>
<td>Trying to save as an alias (beginning with &quot;.&quot;) or as a built-in configuration (beginning with &quot;.&quot;).</td>
</tr>
<tr>
<td>inetErrParamsInvalid</td>
<td>Invalid index</td>
</tr>
<tr>
<td>inetErrConfigCantDelete</td>
<td>Configuration to be renamed is either an alias or a built-in configuration</td>
</tr>
</tbody>
</table>

**Comments** After renaming, the configuration index stays the same so that applications that are set up to use that configuration will still work.
correctly. Note that built-in configuration aliases (ones that start with a period) cannot be renamed.

**Compatibility**
Implemented only if [Wireless Internet Feature Set](#) is present.

### INetLibConfigSaveAs

**Purpose**
Saves the current network configuration settings under the given name.

**Declared In**
INetMgr.h

**Prototype**
Err INetLibConfigSaveAs (UInt16 refNum, MemHandle inetH, INetConfigNamePtr nameP)

**Parameters**
- *libRefnum* Refnum of the Internet library.
- *inetH* Inet handle allocated by INetLibOpen.
- *nameP* Pointer to an [INetConfigNameType](#) structure holding the name of the configuration. The name cannot start with a period or an underscore.

**Result**
- 0 No error
- inetErrConfigBadName Trying to save as an alias (beginning with ".") or as a built-in configuration (beginning with ".")
- inetErrConfigTooMany The internal configurations table is full. No more entries can be stored.

**Comments**
If the configuration name specified already exists, it is replaced with the new settings.

**Compatibility**
Implemented only if [Wireless Internet Feature Set](#) is present.
INetLibGetEvent

Purpose  A replacement for EvtGetEvent that informs an application of status changes to Internet sockets as well as user interface events.

Declared In  INetMgr.h

Prototype  void INetLibGetEvent (UInt16  libRefnum, MemHandle inetH, INetEventType *eventP, Int32 timeout)

Parameters  
- `libRefnum`  Refnum of the Internet library.
- `inetH`  Inet handle allocated by INetLibOpen, or NULL.
- `eventP`  The event structure is returned via this pointer.
- `timeout`  Timeout in ticks. Specify evtWaitForever to wait forever.

Result  
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error</td>
</tr>
</tbody>
</table>

Comments  This call is designed to replace EvtGetEvent in applications which use the Internet library. For convenience, if `inetH` is NULL, INetLibGetEvent is equivalent to EvtGetEvent.

INetLibGetEvent returns two additional events besides those returned by EvtGetEvent: inetSockReadyEvent and inetSockStatusChangeEvent.

Compatibility  Implemented only if Wireless Internet Feature Set is present.

See Also  INetLibSockStatus, INetLibURLOpen, INetLibSockOpen, INetLibSockRead
**INetLibOpen**

**Purpose** Creates a new application inet handle structure. Opens or increments the open count of the net library.

**Declared In** INetMgr.h

**Prototype**

```
Err INetLibOpen (UInt16 libRefnum, UInt16 config,
                UInt32 flags, DmOpenRef cacheRef,
                UInt32 cacheSize, MemHandle *inetHP)
```

**Parameters**

- `libRefnum` Refnum of the Internet library. Pass the value "INet.lib" to `SysLibFind` to return this refnum.
- `config` Indicates the type of network service desired by the application. Returned by `INetLibConfigIndexFromName`.
- `flags` Currently unused; set to 0.
- `cacheRef` Document cache database reference. Obtain this by using one of the `DmOpenDatabase`... calls. Pass NULL if you don’t want to use a cache.
- `cacheSize` Maximum size of the document cache (in bytes). This is ignored if you pass NULL for cacheRef.
- `inetHP` Pointer to a handle variable.

**Result**

- 0 No error
- `inetErrTooManyClients` Too many clients opened already
- `inetErrIncompatibleInterface` The net library is already open with an incompatible interface
Comments
This call must be made by an application before it can use any other Internet library calls. This call opens the Internet library and returns a pointer to an inet handle, which is then passed to subsequent calls to the Internet library. Every application that opens the Internet library gets its own unique inet handle.

When an application is done using the Internet library, it must call INetLibClose, which closes both the Internet library and the net library, if necessary.

Compatibility
Implemented only if Wireless Internet Feature Set is present.

See Also
INetLibClose, INetLibConfigIndexFromName

INetLibSettingGet

Purpose
Retrieves current settings for an inet handle.

Declared In
INetMgr.h

Prototype
Err INetLibSettingGet (UInt16 libRefnum, MemHandle inetH, UInt16 setting, void *bufP, UInt16 *bufLenP)

Parameters
- libRefnum
  Refnum of the Internet library.
- inetH
  Inet handle allocated by INetLibOpen.
- setting
  The setting to get. Specify one of the INetSettingEnum enumerated types.
- bufP
  Pointer to buffer where the return value is to be put.
- bufLenP
  Size of bufP on entry. Size of setting value on exit.

Result
0
No error
**INetLibSettingSet**

**Purpose**
Changes a setting for an inet handle.

**Declared In**
INetMgr.h

**Prototype**
Err INetLibSettingSet (UInt16 libRefnum, 
MemHandle inetH, UInt16 setting, void *bufP, 
UInt16 bufLen)

**Parameters**
- `libRefnum` Refnum of the Internet library.
- `inetH` Inet handle allocated by INetLibOpen.
- `setting` The setting to set. Specify one of the INetSettingEnum enumerated types.
- `bufP` Pointer to the new setting value.
- `bufLen` Size of the value in bufP.

**Result**
- 0 No error

**Comments**
This call can be used to retrieve the current settings of an inet handle. Some settings have default values that are stored in the system preferences database; see INetSettingEnum for details.

**Compatibility**
Implemented only if Wireless Internet Feature Set is present.

**See Also**
INetLibOpen, INetLibSettingSet, INetSettingEnum
inetErrParamsInvalid  Invalid setting specified
inetErrSettingSizeInvalid  bufLen is the incorrect size for the specified setting

Comments  Any changes made to the settings last only as long as the inetH is around (until INetLibClose is called) and do not affect other applications that might be using the Internet library.

An important note is that settings made through this call essentially change the default values for any sockets subsequently created through INetLibURLOpen or INetLibSockOpen.

Compatibility  Implemented only if Wireless Internet Feature Set is present.

See Also  INetLibSettingGet, INetSettingEnum

INetLibSockClose

Purpose  Closes an inet socket handle.

Declared In  INetMgr.h

Prototype  Err INetLibSockClose (UInt16 libRefnum, MemHandle socketH)

Parameters  -> libRefnum  Refnum of the Internet library.
            -> socketH  Handle of the socket to close.

Result  0  No error

Comments  This call closes down and releases all memory associated with a socket created by INetLibSockOpen or INetLibURLOpen.
Compatibility     Implemented only if Wireless Internet Feature Set is present.

See Also     INetLibOpen, INetLibSockOpen, INetLibURLOpen

INetLibSockConnect

Purpose     Establishes a connection with a remote host.

Declared In     INetMgr.h

Prototype     Err INetLibSockConnect (UInt16 libRefnum, MemHandle sockH, UInt8 *hostnameP, UInt16 port, Int32 timeou)

Parameters

- `libRefnum`     Refnum of the Internet library.
- `sockH`     Handle (allocated by INetLibSockOpen or INetLibURLOpen) of the socket to connect.
- `hostnameP`     Pointer to host name string; can be dotted decimal text string.
- `port`     Port number, or 0 for default port.
- `timeou`     Timeout in ticks. Specify evtWaitForever to wait forever.

Result

0     No error

Comments     This call associates a remote host name and port number with a socket and, depending on the socket protocol, initiates a connection with that remote host.

This call may return immediately before actually finishing the connect. The application can simply go ahead and submit additional calls such as INetLibSockRead, or it may wait for the connect to complete by either polling INetLibSockStatus until the socket status is inetStatusConnected (not recommended), or by waiting for an inetSockStatusChangeEvent event from INetLibGetEvent and checking the status then (preferred).
Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibSockOpen, INetLibSockStatus, INetLibGetEvent

**INetLibSockHTTPAttrGet**

**Purpose** Queries HTTP request header formed by the local host, or the response header information returned by a remote host.

**Declared In** INetMgr.h

**Prototype**

```
Err INetLibSockHTTPAttrGet (UInt16 libRefnum, MemHandle sockH, UInt16 attr, UInt16 attrIndex, void *bufP, UInt32 *bufLenP)
```

**Parameters**

- `-> libRefnum` Refnum of the Internet library.
- `-> sockH` Handle (allocated by INetLibSockOpen or INetLibURLOpen) of the socket.
- `-> attr` The attribute to get. Specify one of the INetHTTPAttrEnum values.
- `-> attrIndex` The attribute index (if any). Currently unused.
- `<- bufP` Pointer to the address where the result is returned.
- `<- bufLenP` Pointer to the size of bufP on entry; size of returned value on exit.

**Result**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>inetErrSettingNotImplemented</td>
<td>Invalid setting specified</td>
</tr>
<tr>
<td>inetErrSettingSizeInvalid</td>
<td>bufLen is the incorrect size for the specified setting</td>
</tr>
</tbody>
</table>
Internet Library
Internet Library Functions

Comments
This call queries either the request header formed by
INetLibSockHTTPReqCreate and INetLibSockHTTPAttrSet,
or the response header returned by the remote host.

Compatibility
Implemented only if Wireless Internet Feature Set is present.

See Also
INetLibSockHTTPReqCreate

INetLibSockHTTPAttrSet

Purpose
Adds additional HTTP request headers to an HTTP request in a
socket.

Declared In
INetMgr.h

Prototype
Err INetLibSockHTTPAttrSet (UInt16 libRefnum,
MemHandle sockH, UInt16 attr, UInt16 attrIndex,
UInt8 *bufP, UInt16 bufLen, UInt16 flags)

Parameters
-> libRefnum Refnum of the Internet library.
-> sockH Handle (allocated by INetLibSockOpen
or INetLibURLOpen) of the socket.
-> attr The attribute to set. Specify one of the
INetHttpRequestEnum values.
-> attrIndex The attribute index (if any). Currently unused.
-> bufP Pointer to additional header text to add.
-> bufLen Length of bufP.
-> flags Flags that control the addition of new headers.
Currently unused.

Result
0 No error
This call modifies attributes of an HTTP request formed by INetLibSockHTTPReqCreate. Generally, attributes are set only before calling INetLibSockHTTPReqSend.

**Compatibility**
Implemented only if Wireless Internet Feature Set is present.

**See Also**
INetLibSockHTTPReqCreate, INetLibSockHTTPReqSend

**INetLibSockHTTPReqCreate**

**Purpose**
Forms an HTTP request for the socket.

**Declared In**
INetMgr.h

**Prototype**
Err INetLibSockHTTPReqCreate (UInt16 libRefnum, MemHandle sockH, UInt8 *verbP, UInt8 *resNameP, UInt8 *refererP)

**Parameters**
- `libRefnum` Refnum of the Internet library.
- `sockH` Handle (allocated by INetLibSockOpen or INetLibURLOpen) of the socket.
- `verbP` Reserved for future use.
- `resNameP` Pointer to a string holding the name of the resource to get or put.
- `refererP` Pointer to a string holding the name of the referring URL, or NULL if none.

**Result**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>inetErrParamsInvalid</td>
<td>Not an HTTP socket</td>
</tr>
</tbody>
</table>
Comments
This call forms an HTTP request for the socket. The request is not actually sent to the remote host until INetLibSockHTTPReqSend is called. The HTTP verb used in the request is determined by the value of the writeP parameter passed to INetLibSockHTTPReqSend: if this parameter is NULL, “GET” is used. Otherwise, “POST” is used.

After a call to INetLibSockHTTPReqCreate but before a call to INetLibSockHTTPReqSend, the application can add additional HTTP request headers using INetLibSockHTTPAttrSet.

Compatibility
Implemented only if Wireless Internet Feature Set is present.

See Also
INetLibSockHTTPAttrSet, INetLibSockHTTPReqSend

INetLibSockHTTPReqSend

Purpose
Sends an HTTP request to the remote host or looks for data in the cache.

Declared In
INetMgr.h

Prototype
Err INetLibSockHTTPReqSend (UInt16 libRefnum, MemHandle sockH, void *writeP, Uint32 writeLen, Int32 timeout)

Parameters
- > libRefnum  Refnum of the Internet library.
- > sockH  Handle (allocated by INetLibSockOpen or INetLibURLOpen) of the socket.
- > writeP  Pointer to additional data to send after the request headers. Usually used for POST and PUT operations.
- > writeLen  Number of bytes in writeP.
Internet Library
Internet Library Functions

INetLibSockHTTPReqSend

-> timeout  Timeout in ticks.

Result

0  No error

inetErrRequestTooLong  Request too big

inetErrEncryptionNotAvail  Encryption requested but not available

Comments

This call sends an HTTP request created by
INetLibSockHTTPReqCreate and INetLibSockHTTPAttrSet
to the remote host. If this is an POST or PUT operation, the data to
write can be specified in writeP.

INetLibSockHTTPReqSend doesn’t always do network I/O. If
the proper socket flag is set, it checks first to see if the requested
data is already in the cache. If it is, then a pointer to the cached data
is stored in the socket and the socket status is updated to show that
data is ready to be read. This will trigger an
inetSockReadyEvent event.

The socket flag (inetOpenURLFlagLookInCache) that causes the
cache to be checked first can be set via the flags parameter to
INetLibURLOpen or by calling INetLibSockSettingSet with
the appropriate setting (inetSockSettingFlags).

After sending the request, the application can wait for the response
to arrive by either polling INetLibSockStatus until the
inputReady boolean is set (not recommended), or by waiting for
an inetSockReadyEvent event from INetLibGetEvent
(preferred).

Compatibility

Implemented only if Wireless Internet Feature Set is present.

See Also

INetLibSockHTTPReqCreate, INetLibSockHTTPAttrSet,
INetLibGetEvent
**INetLibSockOpen**

**Purpose**  Creates and returns a new inet socket handle.

**Declared In**  INetMgr.h

**Prototype**  
```c
Err INetLibSockOpen (UInt16 libRefnum, MemHandle inetH, UInt16 scheme, MemHandle *sockHP)
```

**Parameters**
- `libRefnum`  Refnum of the Internet library.
- `inetH`  Inet handle allocated by INetLibOpen.
- `scheme`  The protocol scheme to use. Specify one of the INetSchemeEnum types.
- `sockHP`  Pointer to the address where the socket handle is returned.

**Result**

<table>
<thead>
<tr>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>inetErrTooManySockets</td>
</tr>
<tr>
<td>inetErrSchemeNotSupported</td>
</tr>
</tbody>
</table>

**Comments**

This call creates a new socket for the given scheme. No network I/O is performed. This is a relatively low level call that can be used in place of INetLibURLOpen when finer control over the socket settings is required.

Using INetLibURLOpen, an HTTP request can be handled with the simple sequence: INetLibURLOpen, INetLibSockRead, and INetLibSockClose. When using INetLibSockOpen, the same HTTP request would be handled by replacing the INetLibURLOpen call with the sequence: INetLibSockOpen, INetLibSockSettingSet (optional), INetLibSockConnect, INetLibSockHTTPReqCreate, INetLibSockHTTPAttrSet (optional), and INetLibSockHTTPReqSend.
The use of INetLibSockOpen allows an application finer control over the socket settings (by calling INetLibSockSettingSet) and the HTTP request headers (by calling INetLibSockHTTPAttrSet).

**Compatibility**
Implemented only if Wireless Internet Feature Set is present.

**See Also**
INetLibOpen, INetLibURLOpen, INetLibSockRead,
INetLibSockClose, INetLibSockSettingSet,
INetLibSockHTTPAttrSet

### INetLibSockRead

**Purpose**
Reads data from a socket.

**Declared In**
INetMgr.h

**Prototype**
Err INetLibSockRead (UInt16 libRefnum,
MemHandle sockH, void *bufP, UInt32 reqBytes,
UInt32 *actBytesP, Int32 timeout)

**Parameters**
- `libRefnum` Refnum of the Internet library.
- `sockH` Inet handle allocated by INetLibOpen.
- `bufP` Pointer to buffer where the data is placed.
- `reqBytes` Requested number of bytes.
- `actBytesP` Pointer to the actual number of bytes read.
- `timeout` Timeout in ticks. Specify evtWaitForever to wait forever.

**Result**
0 No error

**Comments**
This call attempts to read `reqBytes` bytes from the given socket. It returns the actual number of bytes read in `actBytesP`. If the connection with the remote host has been closed, `actBytesP` contains 0 on exit.
Note that it is normal for the actual bytes read to be less than the requested number of bytes. The application should be prepared to call this routine repeatedly until the desired number of bytes have been read or until *actBytesP contains 0, indicating the connection has been closed, or until an error is returned.

This call returns as much data as possible without blocking, however, if no data is available to be read, it does block until at least one byte is available.

Normally, applications will wait for an inetSockReadyEvent from INetLibGetEvent before calling INetLibSockRead. Alternatively, the application could call INetLibSockStatus to determine if the socket has any data ready (not recommended), or could simply rely on INetLibSockRead to block until at least one byte is available to read. If no data is available before the timeout expires, inetErrReadTimeout error is returned.

**Compatibility**

Implemented only if Wireless Internet Feature Set is present.

**See Also**

INetLibURLOpen, INetLibSockOpen, INetLibSockStatus, INetLibSockClose, INetLibGetEvent

### INetLibSockSettingGet

**Purpose**

Retrieves a socket setting.

**Declared In**

INetMgr.h

**Prototype**

Err INetLibSockSettingGet (UInt16 libRefnum, MemHandle socketH, UInt16 setting, void *bufP, UInt16 *bufLenP)

**Parameters**

- `-> libRefnum` Refnum of the Internet library.
- `-> socketH` Handle (allocated by INetLibSockOpen or INetLibURLOpen) of the socket to get a setting from.
- `-> setting` The setting to get. Specify one of the INetSockSettingEnum values.
InetLibSockSettingSet

Purpose
Changes a setting of a socket.

Declared In
INetMgr.h

Prototype
Err INetLibSockSettingSet (UInt16 libRefnum, MemHandle socketH, UInt16 setting, void *bufP, UInt16 bufLen)

Parameters
- -> libRefnum
  Refnum of the Internet library.
- -> socketH
  Handle (allocated by INetLibSockOpen or INetLibURLOpen) of the socket to set.
- -> setting
  The setting to set. Specify one of the INetSockSettingEnum values.
- -> bufP
  Pointer to buffer containing the new setting value.

Result
0
No error
inetErrParamsInvalid
Invalid setting requested
inetErrSettingSizeInvalid
*bufLenP is the incorrect size for the requested setting

Compatibility
Implemented only if Wireless Internet Feature Set is present.

See Also
INetLibSockSettingSet
INetLibSockSetting

Purpose
Retrieves the current status of a socket.

Declared In
INetMgr.h

Prototype
Err INetLibSockSetting (UInt16 libRefnum,
MemHandle socketH, UInt16 *statusP,
Err *sockErrP, Boolean *inputReadyP,
Boolean *outputReadyP)

Parameters
-> libRefnum  Refnum of the Internet library.
-> socketH    Handle (allocated by INetLibSockOpen or INetLibURLOpen) of the socket to get status on.
<- statusP    Pointer to the address where the status is returned. The status will be one of the INetStatusEnum values.
<- sockErrP   Currently unused.

Result
0 No error
inetErrSettingNotImplemented Invalid setting specified
inetErrSettingSizeInvalid bufLen is the incorrect size for the setting

Comments
This call can be use to override a general setting for a particular socket.

Compatibility
Implemented only if Wireless Internet Feature Set is present.

See Also
INetLibSockSettingGet

INetLibSockStatus

Purpose
Retrieves the current status of a socket.

Declared In
INetMgr.h

Prototype
Err INetLibSockStatus (UInt16 libRefnum,
MemHandle socketH, UInt16 *statusP,
Err *sockErrP, Boolean *inputReadyP,
Boolean *outputReadyP)

Parameters
-> libRefnum  Refnum of the Internet library.
-> socketH    Handle (allocated by INetLibSockOpen or INetLibURLOpen) of the socket to get status on.
<- statusP    Pointer to the address where the status is returned. The status will be one of the INetStatusEnum values.
<- sockErrP   Currently unused.

Result
0 No error
inetErrSettingNotImplemented Invalid setting specified
inetErrSettingSizeInvalid bufLen is the incorrect size for the setting

Comments
This call can be use to override a general setting for a particular socket.

Compatibility
Implemented only if Wireless Internet Feature Set is present.

See Also
INetLibSockSettingGet
<- inputReadyP  Pointer to a Boolean; true is returned if the socket has data available to read.

<- outputReadyP  Pointer to a Boolean; true is returned if the socket can accept data for writing.

Result

0  No error

Comments  Most applications that use INetLibGetEvent will rarely need to use this call since socket status changes are returned in the event structure.

Compatibility  Implemented only if Wireless Internet Feature Set is present.

See Also  INetLibURLOpen, INetLibSockOpen, INetLibSockRead, INetLibGetEvent

INetLibURLCrack

Purpose  Cracks a URL text string into its components.

Declared In  INetMgr.h

Prototype  Err INetLibURLCrack (UInt16 libRefnum, UInt8 *urlTextP, INetURLType *urlP)

Parameters  -> libRefnum  Refnum of the Internet library.
-> urlTextP  Pointer to URL text string.
Internet Library
Internet Library Functions

**Internet Library Functions**

```c
<-> urlP  // Pointer to address where the URL information block is to be returned.
```

### Result

- **0**  // No error
- **inetErrParamsInvalid**  // urlTextP is NULL or empty, or urlP is NULL
- **inetErrURLVersionInvalid**  // urlP is wrong version

### Comments

If a pointer member of `urlP` is set to `NULL` on entry, then on exit it will point to the start of that component within the original `urlTextP` string; the associated member length is set to the length of that URL component. If a pointer member of `urlP` is not `NULL` on entry, then it must point to a buffer of sufficient size to hold the member data, and on exit the component string will be copied into this buffer and the associated member length will be updated with the actual size. Note that the returned strings are not null-terminated, so the length values are important.

It’s easiest to initialize the `InetURLType` block to zeros and let this function fill in all the information about the URL components.

The `InetURLType` block returned from this function has the following structure:

```c
typedef struct {
    UInt16 version; // 0, for future compatibility
    UInt8 *schemeP; // ptr to scheme portion
    UInt16 schemeLen; // size of scheme portion
    UInt16 schemeEnum; // INetSchemeEnum; the scheme
    UInt8 *usernameP; // ptr to username portion
    UInt16 usernameLen; // size of username
    UInt8 *passwordP; // ptr to password portion
    UInt16 passwordLen; // size of password
    UInt8 *hostnameP; // ptr to host name portion
    UInt16 hostnameLen; // size of host name
    UInt16 port; // port number
    UInt8 *pathP; // ptr to path portion
    UInt16 pathLen; // size of path
} InetURLType;
```

*Palm OS Programmer’s API Reference* 1883
Internet Library
Internet Library Functions

UInt8 *paramP; // ptr to param (;param)
UInt16 paramLen; // size of param
UInt8 *queryP; // ptr to query (?query)
UInt16 queryLen; // size of query
UInt8 *fragP; // ptr to fragment (#frag)
UInt16 fragLen; // size of fragment
}

INetURLType

Compatibility
Implemented only if Wireless Internet Feature Set is present.

INetLibURLGetInfo

Purpose
Returns information about a URL.

Declared In
INetMgr.h

Prototype
Err INetLibURLGetInfo (UInt16 libRefnum,
MemHandle inetH, UInt8 *urlTextP,
INetURLInfoType *urlInfoP)

Parameters
- libRefnum Refnum of the Internet library.
- inetH Inet handle allocated by INetLibOpen.
- urlTextP Pointer to URL text string.
<-> urlInfoP Pointer to address where the URL information structure is to be returned.

Result
0 No error
inetErrParamsInvalid urlInfoP is NULL or incorrect version

Comments
The InetURLInfo block returned from this function has the following structure:

typedef struct {
UInt16 version; // 0, for future compatibility
UInt16 flags; // flags word
} INetURLInfo
(UInt32 undefined; // reserved for future use
} INetURLInfo

The flags word can consist of some combination of these values:

inetURLInfoFlagIsSecure // URL was encrypted
inetURLInfoFlagIsRemote // URL was retrieved from the net
inetURLInfoFlagIsInCache // URL is stored in the cache

Compatibility Implemented only if Wireless Internet Feature Set is present.

**INetLibURLOpen**

**Purpose** Accesses a URL on the Internet or in the cache.

**Declared In** INetMgr.h

**Prototype**

Err INetLibURLOpen (UInt16 libRefnum,
MemHandle inetH, UInt8 *urlP,
UInt8 *cacheIndexURLP, MemHandle *sockHP,
Int32 timeout, UInt16 flags)

**Parameters**

- **-> libRefnum** Refnum of the Internet library.
- **-> inetH** Inet handle allocated by INetLibOpen.
- **-> urlP** Pointer to string containing the URL to access.
- **-> cacheIndexURLP** Pointer to URL string under which the requested URL should be indexed in the cache. Specify NULL if you don’t need to index the cache. If you are using the Web Clipping Application Viewer cache (not recommended), you must follow the Viewer convention, which is to pass the URL of the active PQA.
- **<- sockHP** Pointer to address where the socket handle is returned.
Internet Library
Internet Library Functions

-> timeout
Timeout in ticks. Specify \texttt{evtWaitForever} to wait forever.

-> flags
Flags indicating caching and encryption options desired. Specify zero, one, or more of the URL open flags (see \texttt{URL Open Constants}).

Result

0 \hspace{1cm} \text{No error}

\texttt{inetErrParamsInvalid} \hspace{1cm} \texttt{urlP} is NULL

Comments

This call sets up a connection to a resource on the Internet addressed by \texttt{urlP} and returns a socket handle. Note that if you specify that the cache should be searched first, and if the data is found in the cache, no network I/O occurs. The application can then read that socket resource through the \texttt{INetLibSockRead} call.

This call is a convenience routine that internally makes the following calls for \texttt{http} URLs: \texttt{INetLibSockOpen}, \texttt{INetLibSockConnect}, \texttt{INetLibSockHTTPReqCreate}, and \texttt{INetLibSockHTTPReqSend}.

This routine returns immediately before performing any required network I/O. It is then up to the caller to either block on \texttt{INetLibSockRead}, or to use \texttt{INetLibGetEvent} to model asynchronous operation. Using \texttt{INetLibGetEvent} is the preferred way of performing network I/O since it maximizes battery life and user-interface responsiveness.

Compatibility

Implemented only if \texttt{Wireless Internet Feature Set} is present.

See Also

\texttt{INetLibSockOpen, INetLibSockConnect, INetLibSockRead, INetLibSockClose}

\textbf{INetLibURLsAdd}

Purpose

Concatenates two URLs, resulting in one absolute URL.

Declared In

\texttt{INetMgr.h}
Prototype  Err INetLibURLsAdd (UInt16 libRefnum,  
Char *baseURLStr, Char *embeddedURLStr,  
Char *resultURLStr, UInt16 *resultLenP)

Parameters  -> libRefnum    Refnum of the Internet library.  
             -> baseURLStr   Pointer to base URL string.  
             -> embeddedURLStr  
                                Pointer to URL text string to append.  
             <-> resultURLStr  
                                Pointer to resulting URL string.  
             <-> resultLenP    Pointer to size of resultURLStr buffer on entry. On exit, pointer to resulting URL length (including null terminator).

Result  0         No error

Comments  Used to append a URL fragment to a base URL, resulting in an absolute URL string that can be passed to INetLibURLOpen or other functions. This routine ensures that the resulting string conforms to the URL format.

Compatibility  Implemented only if Wireless Internet Feature Set is present.
INetLibWiCmd

Purpose Invokes a command that operates on the wireless indicator.

Declared In INetMgr.h

Prototype Boolean INetLibWiCmd (UInt16 refNum, UInt16 cmd, int enableOrX, int y)

Parameters

- > refNum Refnum of the Internet library.
- > cmd The command to invoke. Specify one of the WiCmdEnum values (see Comments section).
- > enableOrX If cmd is wiCmdSetEnabled, specify 1 to enable the wireless indicator or 0 to disable it. If cmd is wiCmdSetLocation, this specifies the x coordinate of the location.
- > y The y coordinate of the location. Used only if cmd is wiCmdSetLocation.

Result If cmd is wiCmdEnabled, this function returns true if the wireless indicator is enabled or false if it is not. For other command types, the return value is undefined.

Comments

The wireless indicator is a 19x13 pixel image on the screen to indicate the current wireless signal strength. This shows as 0 - 5 bars. If the application is in a non-modal window with a title bar, the preferred location for the indicator is at (140,1).

It automatically updates itself as long as you are calling INetLibGetEvent. It should be shown on screen while a wireless transaction is in progress. It may also be shown when the user has nothing useful to do next but initiate a wireless transaction, and there isn’t much other useful information being displayed.

The WiCmdEnum enum specifies a command that operates on the wireless indicator in the user interface. The definition of this type is found in WirelessIndicator.h and is as follows:

```c
typedef enum {
    wiCmdInit = 0,
    ...
} WiCmdEnum;
```
wiCmdClear,
wiCmdSetEnabled,
wiCmdDraw,
wiCmdEnabled,
wiCmdSetLocation,
wiCmdErase
} WiCmdEnum;

**Value Descriptions**

- **wiCmdInit**: Initializes the wireless indicator. You must invoke this command first, before using any of the others.
- **wiCmdClear**: Applications shouldn't use this command. To erase the indicator, disable it by using wiCmdSetEnabled and passing 0 for enableOrX.
- **wiCmdSetEnabled**: Enables or disables the wireless indicator.
- **wiCmdDraw**: Redraws the wireless indicator using the latest data. Applications don’t need to use this command since the indicator is redrawn automatically by INetLibGetEvent.
- **wiCmdEnabled**: Returns a Boolean indicating if the wireless indicator is enabled.
- **wiCmdSetLocation**: Sets the location for the wireless indicator on the screen.
- **wiCmdErase**: Erases the wireless indicator. Applications shouldn’t use this command. To erase the indicator, disable it by using wiCmdSetEnabled and passing 0 for enableOrX.

**Compatibility**

Implemented only if [Wireless Internet Feature Set](#) is present.
PalmOSGlue Library

This chapter describes the API provided in the link library PalmOSGlue (PalmOSGlue.lib or libPalmOSGlue.a).

You use PalmOSGlue if you want to use the international and text manager features described in the chapter “Localized Applications” on page 363 in the Palm OS Programmer’s Companion, vol. I and you want to maintain backward compatibility with earlier releases of Palm OS®. If you link with PalmOSGlue, include the headers in the folder Incs\Libraries\PalmOSGlue, and make calls as they are listed in this chapter, then your code will run regardless of whether the device’s version of the operating system implements international support. The code in PalmOSGlue either uses the text manager or international manager on the ROM or, if the managers don’t exist, executes a simple Latin equivalent of the function.

NOTE: PalmOSGlue is a link library, not a shared library. Linking with this library increases your application’s code size. The amount by which your code size increases varies depending on the number of library functions you call. Use PalmOSGlue only on versions 2.0 and later of Palm OS.

In addition to covering the text and international manager API, PalmOSGlue adds some functions that are not included in any version of the Palm OS. This chapter describes the functions that are unique to PalmOSGlue and provides a mapping of PalmOSGlue calls to calls that exist in later versions of Palm OS.
PalmOSGlue Functions

The following table shows the mapping between the functions declared in the glue headers and the international functions and macros. To learn more about a glue function, click the link in the right column.

This table lists only those functions that map to a function that exists in newer versions of the OS. The functions that are exclusive to PalmOSGlue are not listed. They are described following this table.

Table 75.1 PalmOSGlue function mappings

<table>
<thead>
<tr>
<th>This PalmOSGlue function...</th>
<th>...is identical to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>BmpGlueGetBitDepth</td>
<td>BmpGetBitDepth</td>
</tr>
<tr>
<td>BmpGlueGetDimensions</td>
<td>BmpGetDimensions</td>
</tr>
<tr>
<td>BmpGlueGetNextBitmap</td>
<td>BmpGetNextBitmap</td>
</tr>
<tr>
<td>DateGlueTemplateToAscii</td>
<td>DateTemplateToAscii</td>
</tr>
<tr>
<td>DateGlueToDOWDMFormat</td>
<td>DateToDOWDMFormat</td>
</tr>
<tr>
<td>FntGlueWCharWidth</td>
<td>FntWCharWidth</td>
</tr>
<tr>
<td>FntGlueWidthToOffset</td>
<td>FntWidthToOffset</td>
</tr>
<tr>
<td>LmGlueGetLocaleSetting</td>
<td>LmGetLocaleSetting</td>
</tr>
<tr>
<td>LmGlueGetNumLocales</td>
<td>LmGetNumLocales</td>
</tr>
<tr>
<td>LmGlueLocaleToIndex</td>
<td>LmLocaleToIndex</td>
</tr>
<tr>
<td>LstGlueGetTopItem</td>
<td>LstGetTopItem</td>
</tr>
<tr>
<td>OmGlueGetCurrentLocale</td>
<td>OmGetCurrentLocale</td>
</tr>
<tr>
<td>OmGlueGetSystemLocale</td>
<td>OmGetSystemLocale</td>
</tr>
<tr>
<td>ResGlueLoadConstant</td>
<td>ResLoadConstant</td>
</tr>
<tr>
<td>SysGlueGetTrapAddress</td>
<td>SysGetTrapAddress</td>
</tr>
<tr>
<td>TsmGlueGetFepMode</td>
<td>TsmGetFepMode</td>
</tr>
<tr>
<td>TsmGlueSetFepMode</td>
<td>TsmSetFepMode</td>
</tr>
</tbody>
</table>
### Table 75.1 PalmOSGlue function mappings (continued)

<table>
<thead>
<tr>
<th>This PalmOSGlue function...</th>
<th>...is identical to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>TblGlueGetNumberOfColumns</td>
<td>TblGetNumberOfColumns</td>
</tr>
<tr>
<td>TblGlueGetTopRow</td>
<td>TblGetTopRow</td>
</tr>
<tr>
<td>TblGlueSetSelection</td>
<td>TblSetSelection</td>
</tr>
<tr>
<td>TxtGlueByteAttr</td>
<td>TxtByteAttr</td>
</tr>
<tr>
<td>TxtGlueCaselessCompare</td>
<td>TxtCaselessCompare</td>
</tr>
<tr>
<td>TxtGlueCharAttr</td>
<td>TxtCharAttr</td>
</tr>
<tr>
<td>TxtGlueCharBounds</td>
<td>TxtCharBounds</td>
</tr>
<tr>
<td>TxtGlueCharEncoding</td>
<td>TxtCharEncoding</td>
</tr>
<tr>
<td>TxtGlueCharIsAlNum</td>
<td>TxtCharIsAlNum</td>
</tr>
<tr>
<td>TxtGlueCharIsAlpha</td>
<td>TxtCharIsAlpha</td>
</tr>
<tr>
<td>TxtGlueCharIsCtrl</td>
<td>TxtCharIsCtrl</td>
</tr>
<tr>
<td>TxtGlueCharIsDelim</td>
<td>TxtCharIsDelim</td>
</tr>
<tr>
<td>TxtGlueCharIsDigit</td>
<td>TxtCharIsDigit</td>
</tr>
<tr>
<td>TxtGlueCharIsGraph</td>
<td>TxtCharIsGraph</td>
</tr>
<tr>
<td>TxtGlueCharIsHex</td>
<td>TxtCharIsHex</td>
</tr>
<tr>
<td>TxtGlueCharIsLower</td>
<td>TxtCharIsLower</td>
</tr>
<tr>
<td>TxtGlueCharIsPrint</td>
<td>TxtCharIsPrint</td>
</tr>
<tr>
<td>TxtGlueCharIsPunct</td>
<td>TxtCharIsPunct</td>
</tr>
<tr>
<td>TxtGlueCharIsSpace</td>
<td>TxtCharIsSpace</td>
</tr>
<tr>
<td>TxtGlueCharIsUpper</td>
<td>TxtCharIsUpper</td>
</tr>
<tr>
<td>TxtGlueCharIsValid</td>
<td>TxtCharIsValid</td>
</tr>
<tr>
<td>TxtGlueCharSize</td>
<td>TxtCharSize</td>
</tr>
<tr>
<td>TxtGlueCharWidth</td>
<td>FntWCharWidth</td>
</tr>
</tbody>
</table>
Table 75.1 PalmOSGlue function mappings *(continued)*

<table>
<thead>
<tr>
<th>This PalmOSGlue function...</th>
<th>...is identical to...</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TxtGlueCharXAttr</code></td>
<td><code>TxtCharXAttr</code></td>
</tr>
<tr>
<td><code>TxtGlueCompare</code></td>
<td><code>TxtCompare</code></td>
</tr>
<tr>
<td><code>TxtGlueEncodingName</code></td>
<td><code>TxtEncodingName</code></td>
</tr>
<tr>
<td><code>TxtGlueGetChar</code></td>
<td><code>TxtGetChar</code></td>
</tr>
<tr>
<td><code>TxtGlueGetNextChar</code></td>
<td><code>TxtGetNextChar</code></td>
</tr>
<tr>
<td><code>TxtGlueGetPreviousChar</code></td>
<td><code>TxtGetPreviousChar</code></td>
</tr>
<tr>
<td><code>TxtGlueGetTruncationOffset</code></td>
<td><code>TxtGetTruncationOffset</code></td>
</tr>
<tr>
<td><code>TxtGlueMaxEncoding</code></td>
<td><code>TxtMaxEncoding</code></td>
</tr>
<tr>
<td><code>TxtGlueNextCharSize</code></td>
<td><code>TxtNextCharSize</code></td>
</tr>
<tr>
<td><code>TxtGlueParamString</code></td>
<td><code>TxtParamString</code></td>
</tr>
<tr>
<td><code>TxtGluePreviousCharSize</code></td>
<td><code>TxtPreviousCharSize</code></td>
</tr>
<tr>
<td><code>TxtGlueReplaceStr</code></td>
<td><code>TxtReplaceStr</code></td>
</tr>
<tr>
<td><code>TxtGlueSetNextChar</code></td>
<td><code>TxtSetNextChar</code></td>
</tr>
<tr>
<td><code>TxtGlueStrEncoding</code></td>
<td><code>TxtStrEncoding</code></td>
</tr>
<tr>
<td><code>TxtGlueTransliterate</code></td>
<td><code>TxtTransliterate</code></td>
</tr>
<tr>
<td><code>TxtGlueWordBounds</code></td>
<td><code>TxtWordBounds</code></td>
</tr>
<tr>
<td><code>WinGlueDrawChar</code></td>
<td><code>WinDrawChar</code></td>
</tr>
<tr>
<td><code>WinGlueDrawTruncChars</code></td>
<td><code>WinDrawTruncChars</code></td>
</tr>
</tbody>
</table>
**BmpGlueGetCompressionType**

**Purpose**
Returns the compression type used for a specified bitmap.

**Declared In**
BmpGlue.h

**Prototype**
```
BitmapCompressionType BmpGlueGetCompressionType (const BitmapType *bitmapP)
```

**Parameters**
- `bitmapP` Pointer to the bitmap.

**Result**
Returns a `BitmapCompressionType` enum value. If `bitmapP` is `NULL`, or the specified bitmap is not compressed, this function returns `BitmapCompressionTypeNone`. If the specified bitmap’s encoding version is less than 2, this function returns `BitmapCompressionTypeScanLine`.

**See Also**
BmpCompress

---

**BmpGlueGetTransparentValue**

**Purpose**
Indicates if a specified bitmap has transparency—if, when the bitmap is drawn, pixels of a certain value won’t be drawn.

**Declared In**
BmpGlue.h

**Prototype**
```
Boolean BmpGlueGetTransparentValue (const BitmapType *bitmapP, 
UInt32 *transparentValueP)
```

**Parameters**
- `bitmapP` Pointer to the bitmap.
The pixel value that isn’t drawn, if the bitmap has transparency. If the value returned by BmpGlueGetTransparentValue is false, *transparentValueP is left unchanged.

**Result**

Returns true if, when drawing the specified bitmap, Palm OS doesn’t draw pixels that have a value equal to that returned in transparentValueP. Returns false if all pixels are drawn.

**Comments**

You can specify the transparent color when you create the bitmap using Constructor.

**See Also** BmpGlueSetTransparentValue

### BmpGlueSetTransparentValue

**Purpose**

Causes pixels of a specified color not to be drawn when the bitmap is drawn.

**Declared In**

BmpGlue.h

**Prototype**

void BmpGlueSetTransparentValue (BitmapType *bitmapP, UInt32 transparentValue)

**Parameters**

-> bitmapP Pointer to the bitmap.

-> transparentValueP The pixel value that isn’t drawn.

**Result**

Returns nothing.

**Comments**

Does nothing if bitmapP is NULL, if bitmapP is an off-screen bitmap, or if transparentValue is invalid given the bitmap’s bit depth.

If the specified bitmap’s encoding version is less than 2, this function updates it to 2.

**See Also** BmpGlueGetTransparentValue
**CtlGlueGetControlStyle**

**Purpose**
Return the type of the control, such as button, slider, and so on.

**Declared In**
CtlGlue.h

**Prototype**
```
ControlStyleType CtlGlueGetControlStyle (const ControlType *ctlP)
```

**Parameters**
- `ctlP` A pointer to a `ControlType` structure.

**Result**
Returns one of the `ControlStyleType` constants.

**Compatibility**
Implemented only in the PalmOSGlue library.

---

**CtlGlueGetFont**

**Purpose**
Gets the font used when drawing a specified control’s label.

**Declared In**
CtlGlue.h

**Prototype**
```
FontID CtlGlueGetFont (const ControlType *ctlP)
```

**Parameters**
- `ctlP` Pointer to the control object.

**Result**
Returns the ID of the font used to draw the control’s label.

**See Also**
CtlGlueSetFont
CtlGlueGetGraphics

Purpose
Gets the bitmaps displayed in place of a specified control’s label.

Declared In
CtlGlue.h

Prototype
void CtlGlueGetGraphics (const ControlType *ctlP, DmResID *bitmapID, DmResID *selectedBitmapID)

Parameters
-> ctlP
Pointer to the control.

<- bitmapID
Resource ID of the bitmap to display when the graphical control is not selected.

<- selectedBitmapID
Resource ID of the bitmap to display when the graphical control is selected, if, when selected, the graphical control should show a different bitmap.

Result
Returns nothing.

Comments
If the specified control is not a graphical control—one that displays a bitmap in place of the text label—*bitmapID and *selectedBitmapID are set to zero.

This function works with any graphical control, including sliders.

Compatibility
Implemented only if 3.5 New Feature Set is present.

See Also
CtlGlueGetFont, CtlSetGraphics
Palm OS Programmer's API Reference

PalmOSGlue Library
PalmOSGlue Functions

CtlGlueNewSliderControl

Purpose
Create a new slider or feedback slider dynamically and install it in the specified form. The newly-created control is marked as a graphical control.

Declared In
CtlGlue.h

Prototype
SliderControlType *CtlGlueNewSliderControl
(void **formPP, UInt16 ID,
ControlStyleType style, DmResID thumbID,
DmResID backgroundID, Coord x, Coord y,
Coord width, Coord height, UInt16 minValue,
UInt16 maxValue, UInt16 pageSize, UInt16 value)

Parameters
- formPP
  Pointer to the pointer to the form in which the new control is installed. This value is not a handle; that is, the formPP value may change if the object moves in memory. In subsequent calls, always use the new formPP value returned by this function.

- ID
  Symbolic ID of the slider.

- style
  Either sliderCtl or feedbackSliderCtl. See ControlStyleType.

- thumbID
  Resource ID of the bitmap to display as the slider thumb. The slider thumb is the knob that the user can drag to change the slider’s value. To use the default thumb bitmap, pass NULL for this parameter.

- backgroundID
  Resource ID of the bitmap to display as the slider background. To use the default background bitmap, pass NULL for this parameter.

- x
  Horizontal coordinate of the upper-left corner of the slider’s boundaries, relative to the window in which it appears.
PalmOSGlue Library
PalmOSGlue Functions

-> y
 Vertical coordinate of the upper-left corner of the slider’s boundaries, relative to the window in which it appears.

-> width
 Width of the slider, expressed in pixels. Valid values are 1–160.

-> height
 Height of the slider, expressed in pixels. Valid values are 1–160.

-> minValue
 Value of the slider when its thumb is all the way to the left.

-> maxValue
 Value of the slider when its thumb is all the way to the right.

-> pageSize
 Amount by which to increase or decrease the slider’s value when the user clicks to the right or left of the thumb.

-> value
 The initial value to display in the slider.

Result
Returns a pointer to the new slider control. See SliderControlType.

Compatibility
Implemented only if 3.5 New Feature Set is present.

See Also
CtlNewSliderControl, CtlNewGraphicControl, CtlNewControl, CtlValidatePointer, FrmRemoveObject

CtlGlueSetFont

Purpose
Sets the font to use when drawing the control’s label.

Declared In
CtlGlue.h

Prototype
void CtlGlueSetFont (ControlType *ctlP, FontID fontID)

Parameters
-> ctlP
 Pointer to the control object.
-> fontID The ID of the font to use when drawing the control’s label.

Result Returns nothing.

See Also CtlGlueSetFont

CtlGlueSetLeftAnchor

Purpose Causes a control’s left bound to be fixed or to float.

Declared In CtlGlue.h

Prototype

void CtlGlueSetLeftAnchor (ControlType *ctlP, Boolean leftAnchor)

Parameters

-> ctlP Pointer to the control.
-> leftAnchor A value of true causes the left bound of the control to be fixed.

Result Returns nothing.

Comments Used by controls that expand and shrink their width when the label is changed.

FldGlueGetLineInfo

Purpose Retrieve the word-wrapping information for a visible line within a field.

Declared In FldGlue.h

Prototype

Boolean FldGlueGetLineInfo
(const FieldType *fldP, UInt16 lineNum, UInt16 *startP, UInt16 *lengthP)

Parameters

-> fldP A pointer to a FieldType structure.
PalmOSGlue Library
PalmOSGlue Functions

-> lineNum The number of the visible line in the field about which you want to retrieve information. Lines are numbered starting at 0.

<- startP The byte offset into the FieldType’s text field of the first character displayed by this line. If the line is blank, start is equal to the length of the field’s text string.

<- lengthP The length in bytes of the portion of the string displayed on this line. If the line is blank, the length is 0.

Result Returns true if startP and endP contain valid values, or false if the field is a single-line field or does not contain a line numbered lineNum.

Compatibility Implemented only in the PalmOSGlue library.

FntGlueGetDefaultFontID

Purpose Return the font ID of a default font.

Declared In FntGlue.h

Prototype FontID FntGlueGetDefaultFontID (FontDefaultType inFontType)

Parameters -> inFontType A FontDefaultType constant specifying one of the system default fonts. This value can be one of the following:

defaultSystemFont
  The default font for the system.

defaultLargeFont
  The default large font.

defaultSmallFont
  The default small font.
defaultBoldFont
The default bold font.

**Result**
Returns the ID of inFontType.

**Comments**
Use this function whenever you need to obtain a font ID for one of the system default fonts. The default fonts (and thus, the IDs for the default fonts) vary depending on the system’s locale. For example, Japanese systems have a different set of default fonts than systems using the Latin character encoding.

Use this function in place of the constants that specify the IDs of default fonts, as shown in the following table.

<table>
<thead>
<tr>
<th>In place of this...</th>
<th>...use FntGlueGetDefaultFontID with this constant...</th>
</tr>
</thead>
<tbody>
<tr>
<td>stdFont</td>
<td>defaultSystemFont (best for displaying text) or:</td>
</tr>
<tr>
<td></td>
<td>defaultSmallFont (if you want a smaller font)</td>
</tr>
<tr>
<td>largeFont</td>
<td>defaultLargeFont</td>
</tr>
<tr>
<td>largeBoldFont</td>
<td>defaultLargeFont</td>
</tr>
<tr>
<td>boldFont</td>
<td>defaultBoldFont</td>
</tr>
</tbody>
</table>

Note that defaultSystemFont and defaultSmallFont might return the same font ID or different font IDs, depending on the system locale.

**Compatibility**
Implemented only in the PalmOSGlue library.

**See Also**
FontSelect, FntGetFont, FntSetFont
FrmGlueGetActiveField

Purpose  Return the active field for a form.

Declared In  FrmGlue.h

Prototype  extern FieldType *FrmGlueGetActiveField
            (const FormType *formP)

Parameters  -> formP  Pointer to the form, or NULL to use the active form.

Result  Returns a pointer to the active field. Returns NULL if there is no active form or field.

FrmGlueGetDefaultButtonID

Purpose  Gets the resource ID of the object on the form defined as the default button.

Declared In  FrmGlue.h

Prototype  UInt16 FrmGlueGetDefaultButtonID
            (const FormType *formP)

Parameters  -> formP  Pointer to the form.

Result  Returns the resource ID of the object defined as the default button.

See Also  FrmDoDialog, FrmGlueSetDefaultButtonID
**FrmGlueGetHelpID**

**Purpose**

Gets the resource ID number of the form’s help resource.

**Declared In**

FrmGlue.h

**Prototype**

`UInt16 FrmGlueGetHelpID (const FormType *formP)`

**Parameters**

- `formP`  
  Pointer to the form.

**Result**

Returns the resource ID number of the form’s help resource. The help resource is a String resource (type 'tSTR').

**See Also**

FrmGlueSetHelpID

---

**FrmGlueGetLabelFont**

**Purpose**

Gets the font used for a particular label that appears on a form.

**Declared In**

FrmGlue.h

**Prototype**

`FontID FrmGlueGetLabelFont (const FormType *formP, Uint16 labelID)`

**Parameters**

- `formP`  
  Pointer to the form.

- `labelID`  
  ID of a label object in the form (the object’s type must be frmLabelObj).

**Result**

Returns a FontID value of 0 if either `labelID` is invalid or if the object indicated by `labelID` has a type other than frmLabelObj. Otherwise, this function returns the ID of the font used for the label.

**See Also**

FrmGetObjectType, FrmGlueSetLabelFont
FrmGlueGetMenuBarID

Purpose  Gets the ID number of the form’s menu bar.

Declared In  FrmGlue.h

Prototype  UInt16 FrmGlueGetMenuBarID
            (const FormType *formP)

Parameters  -> formP  Pointer to the form.

Result  Returns the ID number of the form’s menu bar, or zero if the form
doesn’t have a menu bar.

See Also  FrmSetMenu

FrmGlueGetObjectUsable

Purpose  Returns whether an object in a form has been hidden.

Declared In  FrmGlue.h

Prototype  Boolean
            FrmGlueGetObjectUsable (const FormType *formP,
            UInt16 objIndex)

Parameters  -> formP  A pointer to a FormType structure.
            -> objIndex  The index of the object on the form.

Result  Returns true if the object is usable, meaning that it is considered
part of the user interface. Returns false if the object is not usable.
Objects that are not usable never appear on the screen. The function
FrmHideObject clears an object’s usable bit to hide that the object.

Comments  Implemented only in the PalmOSGlue library.
FrmGlueSetDefaultButtonID

Purpose  
Designates the object on the form that is to act as the default button.

Declared In  
FrmGlue.h

Prototype  
void FrmGlueSetDefaultButtonID (FormType *formP, Uint16 defaultButton)

Parameters  
- `formP`  
Pointer to the form.

- `defaultButton`  
The resource ID of the object on the form that is to be the default button.

Result  
Returns nothing.

See Also  
FrmDoDialog, FrmGlueGetDefaultButtonID

FrmGlueSetHelpID

Purpose  
Designates the String resource that is to act as the form’s help resource.

Declared In  
FrmGlue.h

Prototype  
void FrmGlueSetHelpID (FormType *formP, Uint16 helpRscID)

Parameters  
- `formP`  
Pointer to the form.

- `helpRscID`  
The resource ID of the String resource that is to be the form’s help resource.

Result  
Returns nothing.

See Also  
FrmGlueGetHelpID
FrmGlueSetLabelFont

**Purpose**
Sets the font used for a particular label that appears on a form.

**Declared In**
FrmGlue.h

**Prototype**
```c
void FrmGlueSetLabelFont (FormType *formP, UInt16 labelID, FontID fontID)
```

**Parameters**
- `formP` Pointer to the form.
- `labelID` ID of a label object in the form (the object’s type must be frmLabelObj).
- `fontID` ID of the font to be used for the label.

**Result**
Returns nothing.

**Comments**
This function does nothing if either `labelID` is invalid or if the object indicated by `labelID` has a type other than frmLabelObj.

**See Also**
FrmGetObjectType, FrmGlueGetLabelFont

IntlGlueGetRoutineAddress

**Purpose**
Return the address of a Text Manager function or of its PalmOSGlue equivalent.

**Declared In**
IntlGlue.h

**Prototype**
```c
void *IntlGlueGetRoutineAddress (IntlSelector selector, const void *latinSymbol)
```

**Parameters**
- `selector` One of the routine selectors defined in IntlMgr.h.
The corresponding `TxtLatinfunc` symbol defined in `IntlGlue.h`.

**Result**

Returns the address of the native Palm OS function if it is defined. If the function is not defined, it returns the address of the corresponding PalmOSGlue function.

**Comments**

Use `IntlGlueGetRoutineAddress` for performance reasons. You can use the address that it returns to call the function at that address without having to go through the International Manager’s trap dispatch table. `IntlGlueGetRoutineAddress` is mostly useful for optimizing the performance of Text Manager functions that are called in a tight loop.

To call `IntlGlueGetRoutineAddress`, you must pass both the international trap for the function and the corresponding symbol. For example, to obtain the address of the `TxtGetNextChar` function or of its PalmOSGlue equivalent, you would make this call:

```c
myTxtGetNextChar =
    IntlGlueGetRoutineAddress(intlTxtGetNextChar, TxtLatinGetNextChar);
```

The returned address is only valid while your application stays locked in memory. For this reason, you should only use the returned address up to the point where your application terminates. When the application starts up again, you should call `IntlGlueGetRoutineAddress` again.

**Compatibility**

Implemented only in the PalmOSGlue library.

**See Also**

`IntlGetRoutineAddress`
**LstGlueGetFont**

**Purpose**
Get the font used to draw a list’s text strings.

**Declared In**
LstGlue.h

**Prototype**
FontID LstGlueGetFont (const ListType *listP)

**Parameters**
- listP
  Pointer to the list.

**Result**
Returns the ID of the font used to draw all list text strings.

**See Also**
LstGlueSetFont

**LstGlueGetItemsText**

**Purpose**
Get the text strings that represent the items in a list.

**Declared In**
LstGlue.h

**Prototype**
Char **LstGlueGetItemsText
(const ListType *listP)

**Parameters**
- listP
  Pointer to the list.

**Result**
Returns a pointer to an array of pointers to the text of the list choices.

**See Also**
LstGetSelectionText, LstSetListChoices
**LstGlueSetFont**

**Purpose**
Specify the font to be used to draw a list’s text strings.

**Declared In**
LstGlue.h

**Prototype**
```c
void LstGlueSetFont (ListType *listP, FontID fontID)
```

**Parameters**
- `listP` Pointer to the list.
- `fontID` ID of the font to be used to draw all list text strings.

**Result**
Returns nothing.

**See Also**
LstGlueGetFont

**LstGlueSetIncrementalSearch**

**Purpose**
Enables or disables incremental search for a sorted popup list.

**Declared In**
LstGlue.h

**Prototype**
```c
void LstGlueSetIncrementalSearch (ListType *listP, Boolean incrementalSearch)
```

**Parameters**
- `listP` Pointer to the list.
- `incrementalSearch` Set to `true` to enable incremental search, `false` to disable it.

**Result**
Returns nothing.

**Comments**
If incremental search is enabled, when the list is displayed the user can navigate the list by entering up to five characters. The list will scroll to present the first list item that matches the entered characters. This feature only works for popup lists, and only works
if the list is sorted and the list items are available to the List Manager (that is, you don’t pass NULL to LstSetListChoices).

**SysGlueTrapExists**

**Purpose**
Macro that indicates if a given trap exists on the current system.

**Declared In**
SysGlue.h

**Prototype**
SysGlueTrapExists (trapNum)

**Parameters**
- `-> trapNum` One of the system trap constants.

**Result**
Returns `true` if the current operating system defines the system trap `trapNum`, or `false` if the trap does not exist on that version of the operating system.

**Compatibility**
Implemented only in the PalmOSGlue library.

**TblGlueGetColumnMasked**

**Purpose**
Determines whether a particular table column is masked.

**Declared In**
TblGlue.h

**Prototype**
Boolean TblGlueGetColumnMasked
(const TableType *tableP, Int16 column)

**Parameters**
- `-> tableP` Pointer to the table.
- `-> column` Column number (zero-based).

**Result**
Returns `true` if the column is masked, `false` otherwise.
Comments

If a table cell’s column is masked and the cell’s row is also masked, the table cell is drawn on the screen but is shaded to obscure the information that it contains.

See Also

TblSetColumnMasked

**TxtGlueCharIsVirtual**

Purpose

Return whether a character is a virtual character or not.

Declared In

TxtGlue.h

Prototype

Boolean TxtGlueCharIsVirtual(UInt16 inModifiers, WChar inChar)

Parameters

- `inModifiers` The value passed in the modifiers field of the keyDownEvent.
- `inChar` A character.

Result

Returns true if the character `inChar` is a virtual character, false otherwise.

Comments

Virtual characters are nondisplayable characters that trigger special events in the operating system, such as displaying low battery warnings or displaying the keyboard dialog. Virtual characters should never occur in any data and should never appear on the screen.

Starting in Palm OS 3.1, the command modifier bit is always set in the keyDownEvent for a virtual character because the range for virtual characters overlaps the range for “real” characters that should appear on the screen. Earlier releases of the operating system did not always set the command modifier for virtual characters.

You can use this function to test whether a character is virtual or not. Pass the `chr` and `modifiers` fields exactly as you received them in the keyDownEvent, and this function performs the appropriate check based on the operating system version.
**Compatibility**  
Implemented only in the PalmOSGlue library.

**TxtGlueFindString**

**Purpose**  
Perform a case-insensitive search for a string in another string.

**Declared In**  
TextMgr.h

**Prototype**

```c
Boolean TxtGlueFindString  
(const Char *inSourceStr,  
const Char *inTargetStr, UInt32 *outPos,  
UInt16 *outLength)
```

**Parameters**

- `-> inSourceStr`  
  Pointer to the string to be searched.
- `-> inTargetStr`  
  Prepared version of the string to be found. This string should either be passed directly from the strToFind field in the `sysAppLaunchCmdFind` launch code’s parameter block or it should be prepared using `TxtGluePrepFindString`.
- `<- outPos`  
  Pointer to the offset of the match in inSourceStr.
- `<- outLength`  
  Pointer to the length in bytes of the matching text.

**Result**

Returns true if the function finds `inTargetStr` within `inSourceStr`; false otherwise.

If found, the values pointed to by the `outPos` and `outLength` parameters are set to the starting offset and the length of the matching text. Unlike `TxtFindString`, if the target string is not found, the values pointed to by `outPos` and `outLength` are not necessarily set to 0.

The search that `TxtGlueFindString` performs is locale-dependent. On most ROMs with Latin-based encodings, `TxtGlueFindString` returns true only if the string is at the beginning of a word. On Shift-JIS encoded ROMs,
TxtGlueFindString returns true if the string is located anywhere in the word.

**Comments**

Use this function instead of FindStrInStr to support the global system find facility. This function contains an extra parameter, outLength, to specify the length of the text that matched. Pass this value to FindSaveMatch in the appCustom parameter. Then when your application receives sysAppLaunchCmdGoTo, the matchCustom field contains the length of the matching text. You use the length of matching text to highlight the match within the selected record.

You must make sure that the parameters inSourceStr and inTargetStr point to the start of a valid character. That is, they must point to the first byte of a multi-byte character, or they must point to a single-byte character; if they don’t, results are unpredictable.

**Compatibility**

Implemented only in the PalmOSGlue library.

**See Also**

TxtFindString, TxtCaselessCompare

---

**TxtGlueGetHorizEllipsisChar**

**Purpose**

Return the horizontal ellipsis character.

**Declared In**

TxtGlue.h

**Prototype**

WChar TxtGlueGetHorizEllipsisChar (void)

**Parameters**

None.

**Result**

Returns the character code for horizontal ellipsis.

**Comments**

Versions 3.1 and higher of the Palm OS use different character codes for the horizontal ellipsis character and the numeric space character than earlier versions did. Use TxtGlueGetHorizEllipsisChar to return the appropriate code for horizontal ellipsis regardless of which version of Palm OS your application is run on.
### PalmOSGlue Library

#### PalmOSGlue Functions

---

**TxtGlueGetNumericSpaceChar**

**Purpose**
Return the numeric space character.

**Declared In**
TxtGlue.h

**Prototype**
WChar TxtGlueGetNumericSpaceChar (void)

**Parameters**
None.

**Result**
Returns the character code for numeric space.

**Comments**
Versions 3.1 and higher of the Palm OS use different character codes for the horizontal ellipsis character and the numeric space character than earlier versions did. Use `TxtGlueGetNumericSpaceChar` to return the appropriate code for numeric space regardless of which version of Palm OS your application is run on.

---

**TxtGlueLowerChar**

**Purpose**
Convert a character to lowercase.

**Declared In**
TxtGlue.h

**Prototype**
WChar TxtGlueLowerChar (WChar inChar)

**Parameters**
- `inChar` A character.

**Result**
Returns the character as a lowercase letter.
Comments  This function does not handle the case in which the lowercase version of a character is represented by two or more characters. If you need to handle this situation, call the `TxtGlueLowerStr` function instead of this one.

Compatibility  Implemented only in the PalmOSGlue library.

See Also  `TxtGlueUpperChar`, `TxtGlueLowerStr`, `TxtGlueUpperStr`, `TxtGlueTransliterate`, `TxtTransliterate`, `StrToLower`  

## TxtGlueLowerStr

### Purpose
Convert a string to all lowercase letters.

### Declared In
`TxtGlue.h`

### Prototype
```c
voidTxtGlueLowerStr(Char*iostream,
UInt16inMaxLength)
```

### Parameters
- `ioString`  The string to be converted.
- `inMaxLength`  The size of the buffer that contains the string, excluding the terminating null character.

### Result
Returns in `ioString` the input string with its letters converted to lowercase.

### Comments
Converting a string from uppercase to lowercase letters or vice versa may change the size of the string. For this reason, you should always check the size of the `ioString` after this call returns.

You must make sure that the parameter `ioString` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

This function can only handle characters in the ISO Latin 1 character encoding unless the International Feature Set is present.
#### Compatibility
Implemented only in the PalmOSGlue library.

#### See Also
TxtGlueUpperStr, TxtGlueLowerChar, TxtGlueUpperChar, StrToLower, TxtGlueTransliterate, TxtTransliterate

## TxtGluePrepFindString

### Purpose
Set up for TxtFindString or FindStrInStr.

### Declared In
TxtGlue.h

### Prototype
```c
void TxtGluePrepFindString (const Char* inSource, CharPtr outDest, UInt16 inDstSize)
```

### Parameters
- **-> inSource**
  Pointer to the text to be searched for. Must not be NULL.
- **<- outDest**
  The same text as inSource but converted to a suitable format for searching. outDest must not be the same address as inSource.
- **-> inDstSize**
  The length in bytes of the area pointed to by outDest.

### Result
Returns in outDest an appropriately converted string.

### Comments
Use this function to normalize the string to search for before using TxtGlueFindString, TxtFindString, or FindStrInStr to perform a search that is internal to your application. If you use any of these three search routines in response to the sysAppLaunchCmdFind launch code, the string that the launch code passes in is already properly normalized for the search.

This function normalizes the string to be searched for. The method by which a search string is normalized varies depending on the version of Palm OS and the character encoding supported by the device.

Only inDstSize bytes of inSource are written to outDest. If necessary to prevent overflow of the destination buffer, not all of inSource is converted.
You must make sure that the parameter inSource points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn’t, results are unpredictable.

**Compatibility** Implemented only in the PalmOSGlue library.

### TxtGlueStripSpaces

**Purpose** Strip trailing and/or leading spaces from a string.

**Declared In** TxtGlue.h

**Prototype** 

```c
Char* TxtGlueStripSpaces (Char* ioStr, Boolean leading, Boolean trailing)
```

**Parameters**

- **ioStr** Any string.
- **leading** If true, strip the leading spaces from the string.
- **trailing** If true, strip the trailing spaces from the string.

**Result** Returns `ioStr` with the specified spaces stripped from it. Note that this function both changes the `ioStr` buffer parameter and returns a pointer to it.

**Comments** You must make sure that the parameter `ioStr` points to the start of a a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn’t, results are unpredictable.

**Compatibility** Implemented only in the PalmOSGlue library.
TxtGlueTruncateString

**Purpose**
Determine if a string can be displayed in a given number of pixels. If not, truncate the string.

**Declared In**
TxtGlue.h

**Prototype**
Boolean TxtGlueTruncateString (Char *ioStringP, UInt16 inMaxWidth)

**Parameters**
- `<-> ioStringP` A null-terminated string. Upon return, the string is truncated if necessary so that it can be displayed in `inMaxWidth` pixels.
- `-> inMaxWidth` The maximum width in pixels.

**Result**
Returns `true` if the string was truncated, or `false` if the string can fit without truncation.

**Comments**
This function determines whether `ioStringP` can be displayed in the specified width without being truncated. If it can, `TxtGlueTruncateString` returns `false`. If the string must be truncated, this function truncates the string to one less than the number of characters that can fit in `inMaxWidth` and then appends an ellipsis (…) character. (If the boundary characters are narrower than the ellipsis, more than one character may be dropped to make room). If `inMaxWidth` is narrower than the width of an ellipsis, the string is set to the empty string.

**Compatibility**
Implemented only in the PalmOSGlue library.

**See Also**
FntWidthToOffset, WinDrawTruncChars, TxtGetTruncationOffset
**TxtGlueUpperChar**

**Purpose**  Convert a character to uppercase.

**Declared In**  TxtGlue.h

**Prototype**  WChar TxtGlueUpperChar (WChar inChar)

**Parameters**  -> inChar  Any character.

**Result**  Returns the character as an uppercase letter.

**Comments**  This function does not handle the case in which the uppercase version of a character is represented by two or more characters. If you need to handle this situation, call the **TxtGlueUpperStr** function instead of this one.

**Compatibility**  Implemented only in the PalmOSGlue library.

**See Also**  **TxtGlueLowerChar**, **TxtGlueUpperStr**, **TxtGlueLowerStr**, **TxtGlueTransliterate**, **TxtTransliterate**, **StrToLowerCase**
TxtGlueUpperStr

**Purpose**  Convert a string to all uppercase letters.

**Declared In**  TxtGlue.h

**Prototype**  
```c
void TxtGlueUpperStr (Char* ioString, UInt16 inMaxLength)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&gt; ioString</td>
<td>The string to be converted.</td>
</tr>
<tr>
<td>-&gt; inMaxLength</td>
<td>The size of the buffer that contains the string, excluding the terminating null character.</td>
</tr>
</tbody>
</table>

**Result**  Returns in ioString the input string with its letters converted to uppercase.

**Comments**  Converting a string from uppercase to lowercase letters or vice versa may change the size of the string. For this reason, you should always check the size of the ioString after this call returns.

You must make sure that the parameter ioString points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn’t, results are unpredictable.

This function can only handle characters in the ISO Latin 1 character encoding unless the International Feature Set is present.

**Compatibility**  Implemented only in the PalmOSGlue library.

**See Also**  
- TxtGlueLowerStr, TxtGlueUpperChar, TxtGlueLowerChar
- TxtGlueTransliterate, TxtTransliterate, StrToLower
WinGlueGetFrameType

Purpose          Gets the frame type for a specified window.

Declared In      WinGlue.h

Prototype        FrameType WinGlueGetFrameType
                  (const WinHandle winH)

Parameters       -> winH        The window’s handle.

Result           Returns a FrameType value indicating the window’s frame style.

See Also         WinGlueSetFrameType

WinGlueSetFrameType

Purpose          Sets the type of frame to be used for a specified window.

Declared In      WinGlue.h

Prototype        void WinGlueSetFrameType (WinHandle winH,
                  FrameType frame)

Parameters       -> winH        The window’s handle.
                  -> frame        The style of frame to be used.

Result           Returns nothing.

See Also         WinGlueGetFrameType
Bluetooth Library: General Functions

The Bluetooth library is a shared library that provides a direct interface to the Bluetooth communication capability of the Palm OS. This chapter presents reference material for the security and utility functions of the Bluetooth library API:

Security Functions  Describes security functions that allow the application to manage a database of trusted devices. Trusted devices don’t need to undergo authentication when they reconnect with the local device.

Utility Functions  Describes utility functions and macros that are useful when working with the Bluetooth library. They perform such tasks as converting between host and network byte ordering, and converting between device addresses and strings.

The header file `BtLib.h` declares the Bluetooth library functions and macros. The header file `BtLibTypes.h` declares the data structures that you use with those functions and macros. For more information about using the Bluetooth library, see the Palm OS Programmer’s Companion Supplement: Bluetooth.

Security Functions

The Bluetooth security functions allow the application to manage a database of devices that have been bonded or paired. To understand this database, it is important to understand the difference between pairing and bonding.

Typically, devices must authenticate with each other every time they connect to each other. This process is called pairing. However, the
devices can bond instead. Bonding is similar to pairing except that both devices remember the link key for the connection. If the two devices ever want to connect to each other again, they don’t need to repeat the pairing process.

Information about paired and bonded devices is stored in a device database on the local device. If a remote device has bonded with the local device, its record remains in the database until it is explicitly deleted. If the remote device has paired with the local device but not bonded with it, the record is removed when the connection to the remote device terminates.

Remote devices that have bonded with the local device are also called trusted devices.

### New

**BtLibSecurityFindTrustedDeviceRecord**

**Purpose**
Search the device database for the device with the specified Bluetooth address. Return the index of the corresponding device record in the database.

**Declared In**
BtLib.h

**Prototype**

```c
Err BtLibSecurityFindTrustedDeviceRecord
(UInt16 btLibRefNum,
BtLibDeviceAddressTypePtr addrP, UInt16 *index)
```

**Parameters**

- `-> btLibRefNum`  Reference number for the Bluetooth library.
- `-> addrP`  Bluetooth address of remote device.
- `<- index`  Index of the record.

**Result**

Returns btLibErrNoError if successful. Returns btLibErrNotFound if a record with the specified remote device address could not be found.

**See Also**

- BtLibSecurityRemoveTrustedDeviceRecord
- BtLibSecurityGetTrustedDeviceRecordInfo
New

BtLibSecurityGetTrustedDeviceRecordInfo

Purpose
Get information from a device record in the device database.

Declared In
BtLib.h

Prototype
Err BtLibSecurityGetTrustedDeviceRecordInfo
(UInt16 btLibRefNum, UInt16 index,
BtLibDeviceAddressTypePtr addrP,
Char *nameBuffer, UInt8 *nameBufferSize,
BtLibClassOfDeviceType *cod, Boolean *persistent)

Parameters
-> btLibRefNum Reference number for the Bluetooth library.
-> index Index of the record.
<- addrP Bluetooth address of remote device.
<- nameBuffer Pointer to buffer to store user-friendly name of remote device. You must allocate this buffer. Provide a NULL pointer if the user-friendly name is not needed.

<-> nameBufferSize Size of the nameBuffer buffer on entry. On exit, the size of the name.
<- cod Pointer to a BtLibClassOfDeviceType representing the class of the device. You must allocate this structure. Provide a NULL pointer if the device class is not needed.

<- lastConnected The date since the device last connected. This date is measured in seconds since midnight January 1, 1904. Provide a NULL pointer if the date of last connection is not needed.
Bluetooth Library: General Functions
Security Functions

<- persistent If true, the device is bonded and can connect to the local device without authentication. If false, the device is paired but not bonded—it will need to reauthenticate if it connects again. Provide a NULL pointer if this information is not needed.

Result Returns btLibErrNoError if successful. Returns dmErrIndexOutOfRange if a record with the specified index could not be found.

See Also BtLibSecurityFindTrustedDeviceRecord

New BtLibSecurityNumTrustedDeviceRecords

Purpose Return the number of bonded devices in the device database or return the total number of devices in the device database.

Declared In BtLib.h

Prototype UInt16 BtLibSecurityNumTrustedDeviceRecords (UInt16 btLibRefNum, Boolean persistentOnly)

Parameters -> btLibRefNum Reference number for the Bluetooth library.
-> persistentOnly true to obtain the total number of bonded devices in the database. These are the same devices that appear in the trusted devices list. false to obtain the total number of devices in the device database. This includes the devices that are bonded and the devices that are paired but not bonded.

Result Returns the requested number of device records.

See Also BtLibSecurityFindTrustedDeviceRecord, BtLibSecurityGetTrustedDeviceRecordInfo
New

**BtLibSecurityRemoveTrustedDeviceRecord**

**Purpose**
Remove a device record from the device database.

**Declared In**
BtLib.h

**Prototype**
```
Err BtLibSecurityRemoveTrustedDeviceRecord(
    UInt16 btLibRefNum,
    UInt16 index)
```

**Parameters**
- `btLibRefNum` Reference number for the Bluetooth library.
- `index` Index of the record to remove.

**Result**
Returns `btLibErrNoError` if successful. Returns `dmErrIndexOutOfRange` if a record with the specified index could not be found.

**See Also**
-BtLibSecurityFindTrustedDeviceRecord-

**Utility Functions**

This section describes functions and macros that are useful when working with the Bluetooth library.
New BtLibAddrAToBtd

Purpose Convert an ASCII string a Bluetooth device address in colon-separated form to a 48-bit BtLibDeviceAddressType.

Declared In BtLib.h

Prototype Err BtLibAddrAToBtd (UInt16 btLibRefNum, const Char *addressString, BtLibDeviceAddressType *btDevP)

Parameters
-> btLibRefNum Reference number for the Bluetooth library.
-> addressString String containing ASCII colon-separated Bluetooth device address.
<- btDevP Pointer to a BtLibDeviceAddressType to store the converted device address.

Result Returns btLibErrNoError to indicate that the conversion was successful.

See Also BtLibAddrBtdToA
**New**

### BtLibAddrBtdToA

**Purpose**
Convert 48-bit `BtLibDeviceAddressType` to an ASCII string in colon-separated form.

**Declared In**
`BtLib.h`

**Prototype**
```
Err BtLibAddrBtdToA (UInt16 btLibRefNum,
BtLibDeviceAddressType *btDevP, Char *spaceP,
UInt16 spaceSize)
```

**Parameters**
- `-> btLibRefNum`  Reference number for the Bluetooth library.
- `-> btDevP`  Address of a Bluetooth device. This parameter must not be `NULL`.
- `<- spaceP`  Pointer to a buffer to store the ASCII formatted Bluetooth devices address upon return. This parameter must not be `NULL`.
- `-> spaceSize`  Size of the `spaceP` buffer, in bytes. Must be at least 18.

**Result**
Returns `btLibErrNoError` if successful. Returns `btLibErrParamErr` if
- `btDevP` is `NULL`
- `spaceP` is `NULL`
- `spaceSize` is less than 18, the number of bytes required to store the ASCII formatted address

**See Also**
`BtLibAddrAToBtd`
Bluetooth Library: General Functions
Utility Functions

New BtLibL2CapHToNL

Purpose Macro that converts a 32-bit value from host to L2CAP byte order. L2CAP byte order is little endian.

Declared In BtLib.h

Prototype BtLibL2CapHToNL (value)

Parameters -> value 32-bit value to convert.

Result Returns value in L2CAP byte order.

See Also BtLibL2CapHToNS, BtLibL2CapNToHL, BtLibL2CapNToHS

New BtLibL2CapHToNS

Purpose Macro that converts a 16-bit value from host to L2CAP byte order. L2CAP byte order is little endian.

Declared In BtLib.h

Prototype BtLibL2CapHToNS (value)

Parameters -> value 16-bit value to convert.

Result Returns value in L2CAP byte order.

See Also BtLibL2CapHToNL, BtLibL2CapNToHL, BtLibSdpNToHL
New

BtLibL2CapNToHL

Purpose
Macro that converts a 32-bit value from L2CAP to host byte order. L2CAP byte order is little endian.

Declared In
BtLib.h

Prototype
BtLibL2CapNToHL (value)

Parameters
-> value 32-bit value to convert.

Result
Returns value in host byte order.

See Also
BtLibL2CapNToHS, BtLibL2CapHToNL, BtLibL2CapHToNS

New

BtLibL2CapNToHS

Purpose
Macro that converts a 16-bit value from L2CAP to host byte order. L2CAP byte order is little endian.

Declared In
BtLib.h

Prototype
BtLibL2CapNToHS (value)

Parameters
-> value 16-bit value to convert.

Result
Returns value in host byte order.

See Also
BtLibL2CapNToHL, BtLibL2CapHToNL, BtLibL2CapHToNS
New **BtLibRfCommHToNL**

**Purpose**
Macro that converts a 32-bit value from host to RFCOMM byte order. RFCOMM byte order is big endian.

**Declared In**
BtLib.h

**Prototype**
BtLibRfCommHToNL (value)

**Parameters**
->value 32-bit value to convert

**Result**
Returns value in RFCOMM byte order.

**See Also**
BtLibRfCommHToNS, BtLibRfCommNToHL, BtLibRfCommNToHS

New **BtLibRfCommHToNS**

**Purpose**
Macro that converts a 16-bit value from host to RFCOMM byte order. RFCOMM byte order is big endian.

**Declared In**
BtLib.h

**Prototype**
BtLibRfCommHToNS (value)

**Parameters**
->value 16-bit value to convert

**Result**
Returns value in RFCOMM byte order.

**See Also**
BtLibRfCommHToNL, BtLibRfCommHToNS, BtLibRfCommNToHL
**New** BtLibRfCommNToHL

**Purpose**
Macro that converts a 32-bit value from RFCOMM to host byte order. RFCOMM byte order is big endian.

**Declared In**
BtLib.h

**Prototype**
BtLibRfCommNToHL (value)

**Parameters**
->value 32-bit value to convert

**Result**
Returns value in host byte order.

**See Also**
BtLibRfCommNToHS, BtLibRfCommHToNL, BtLibRfCommHToNS

**New** BtLibRfCommNToHS

**Purpose**
Macro that converts a 16-bit value from RFCOMM to host byte order. RFCOMM byte order is big endian.

**Declared In**
BtLib.h

**Prototype**
BtLibRfCommNToHS (value)

**Parameters**
->value 16-bit value to convert

**Result**
Returns value in host byte order.

**See Also**
BtLibRfCommNToHL, BtLibRfCommHToNS, BtLibRfCommHToNL
New

BtLibSdpHToNL

Purpose
Macro that converts a 32-bit value from host to Service Discovery Protocol (SDP) byte order. SDP byte order is big endian.

Declared In
BtLib.h

Prototype
BtLibSdpHToNL (value)

Parameters
->value 32-bit value to convert

Result
Returns value in SDP byte order.

See Also
BtLibSdpHToNS, BtLibSdpNToHL, BtLibSdpNToHS

New

BtLibSdpHToNS

Purpose
Macro that converts a 16-bit value from host to Service Discovery Protocol (SDP) byte order. SDP byte order is big endian.

Declared In
BtLib.h

Prototype
BtLibSdpHToNS (value)

Parameters
->value 16-bit value to convert

Result
Returns value in SDP byte order.

See Also
BtLibSdpHToNL, BtLibSdpNToHL, BtLibSdpNToHS
**New**  

**BtLibSdpNToHL**

**Purpose**  
Macro that converts a 32-bit value from Service Discovery Protocol (SDP) to host byte order. SDP byte order is big endian.

**Declared In**  
BtLib.h

**Prototype**  
`BtLibSdpNToHL (value)`

**Parameters**  
-> value  
32-bit value to convert

**Result**  
Returns value in host byte order.

**See Also**  
BtLibSdpNToHS, BtLibSdpHToNL, BtLibSdpHToNS

**New**  

**BtLibSdpNToHS**

**Purpose**  
Macro that converts a 16-bit value from Service Discovery Protocol (SDP) to host byte order. SDP byte order is big endian.

**Declared In**  
BtLib.h

**Prototype**  
`BtLibSdpNToHS (value)`

**Parameters**  
-> value  
16-bit value to convert

**Result**  
Returns value in host byte order.

**See Also**  
BtLibSdpNToHL, BtLibSdpHToNL, BtLibSdpHToNS
The management API of the Bluetooth library supports the lower levels of the Bluetooth specification, specifically the radio, baseband, and Link Manager Protocol specifications. This chapter presents reference material for the management API:

**Bluetooth Management Data Structures**

This section lists some of the more important types used by the Bluetooth library management functions.

**Management Callback Events**

This section lists the management callback events. Most of the management functions are asynchronous. In other words, they start operations and return before the operations complete. To signal the application that management operations have completed, the Bluetooth library generates management callback events by calling a callback function.

**Management Event Status Codes**

When a management event is generated, the status field of the associated BtLibManagementEventType provides information about why the event occurred. This section explains what these codes mean.

**Library Management Functions**

This section describes the functions that open and close the shared library.
Management Functions
The management functions handle the lower levels of the Bluetooth specification, specifically the radio, baseband, and Link Manager Protocol specifications. These functions perform tasks that include discovering devices, working with Asynchronous Connectionless (ACL) links and piconets, and maintaining global settings for the Bluetooth library.

Application-Defined Functions
This section describes the callback functions that handle management events.

The header file BtLib.h declares the Bluetooth library functions and macros. The header file BtLibTypes.h declares the data structures that you use with those functions and macros. For more information about using the Bluetooth library, see the Palm OS Programmer’s Companion Supplement: Bluetooth.

Bluetooth Management Data Structures
This section lists some of the more important types used by the Bluetooth library management functions.

New BtLibAccessibleModeEnum
The BtLibAccessibleModeEnum enum specifies a device’s accessibility modes. See the “Generic Access Profile” chapter of the Specification of the Bluetooth System for more information about accessibility.

typedef enum {
    btLibNotAccessible = 0x00,
    btLibConnectableOnly = 0x02,
    btLibDiscoverableAndConnectable = 0x03
} BtLibAccessibleModeEnum;
Value Descriptions

**btLibConnectableOnly**

The device responds to a page but not an inquiry.

**btLibDiscoverableAndConnectable**

The device responds to both a page and an inquiry.

**btLibNotAccessible**

The device does not respond to a page or an inquiry.

---

**New**

**BtLibClassOfDeviceType**

The `BtLibClassOfDeviceType` type represents the class of the device and the services it supports.

```c
typedef UInt32 BtLibClassOfDeviceType;
```

A device can support multiple services but only belongs to a single class. The class is specified in two parts: the major class, which broadly classifies the type of device, and the minor class, which together with the major class specifies the type of device in more detail.

An example is a simple cellular telephone. It provides Telephony services. Its major device class is Phone, and its minor device class is Cellular.

The *Bluetooth Assigned Numbers* specification defines a device class as having three bit fields. One field specifies the major service classes supported by the device. Another field specifies the major device class. The third field specifies the minor device class.

The constants provided here allow you to construct a device class that conforms to the Bluetooth specification. You simply perform a logical OR of the constants representing the service classes the device supports, the constant representing the device’s major class, and the constant representing the device’s minor class.

For example, device class of the simple cellular telephone can be computed as follows:
cellPhoneCOD = btLibCOD_Telephony | 
   btLibCOD_Major_Phone | 
   BtLibCOD_Minor_Phone_Cellular;

Constants are also provided to mask the individual bit fields in a device class.

**Major Service Classes**

These constants define the Bluetooth major service classes. The service classes are described in the *Specification of the Bluetooth System*.

```
bLibCOD_LimitedDiscoverableMode
btLibCOD_Networking
btLibCOD_Rendering
btLibCOD_Capturing
btLibCOD_ObjectTransfer
btLibCOD_Audio
btLibCOD_Telephony
btLibCOD_Information
```

**Major Device Classes**

These constants define the Bluetooth major device classes. The major device classes are described in the *Specification of the Bluetooth System*.

```
bLibCOD_Major_Misc
btLibCOD_Major_Computer
btLibCOD_Major_Phone
btLibCOD_Major_Lan_Access_Point
btLibCOD_Major_Audio
btLibCOD_Major_Peripheral
btLibCOD_Major_Unclassified
```

**Computer Minor Device Classes**

These constants define the minor device classes associated with the computer major class. They are described in the *Bluetooth Assigned Numbers* specification.

```
bLibCOD_Minor_Comp_Unclassified
btLibCOD_Minor_Comp_Desktop
btLibCOD_Minor_Comp_Server
```
Phone Minor Device Classes
These constants define the minor device classes are associated with the computer major class. They are described in the Bluetooth Assigned Numbers specification.

- btLibCOD_Minor_Computer_Laptop
- btLibCOD_Minor_Computer_Handheld
- btLibCOD_Minor_Computer_Palm

LAN Access Point Minor Device Classes
These constants define load factors for the LAN access point major device class. LAN access point load factors are described in more detail in the Bluetooth Assigned Numbers specification.

- btLibCOD_Minor_Lan_0
  - Fully available
- btLibCOD_Minor_Lan_17
  - 1-17% utilized
- btLibCOD_Minor_Lan_33
  - 17-33% utilized
- btLibCOD_Minor_Lan_50
  - 33-50% utilized
- btLibCOD_Minor_Lan_67
  - 50-67% utilized
- btLibCOD_Minor_Lan_83
  - 67-83% utilized
- btLibCOD_Minor_Lan_99
  - 83-99% utilized
- btLibCOD_Minor_Lan_NoService
  - Fully utilized
Audio Minor Device Classes

These constants define the minor classes associated with the audio major class. They are described in more detail in the Bluetooth Assigned Numbers specification.

- btLibCOD_Minor_Audio_Unclassified
- btLibCOD_Minor_Audio_Headset

Masks

These constants define bit masks to isolate certain fields of the device class.

- btLibCOD_Service_Mask
  A mask to isolate the major service class field from the other fields of the device class.

- btLibCOD_Major_Mask
  A mask to isolate the major device class field from the other fields of the device class.

- btLibCOD_Minor_Mask
  A mask to isolate the minor device class field from the other fields of the device class.

- btLibCOD_ServiceAny
  Used as a device filter for the BtLibDiscoverMultipleDevices and BtLibDiscoverSingleDevice functions. With this filter, devices providing any service appear in the device list. Same as btLibCOD_Service_Mask.

- btLibCOD_Major_Any
  Used as a device filter for the BtLibDiscoverMultipleDevices and BtLibDiscoverSingleDevice functions. With this filter, devices in any major device class appear in the device list. Same as btLibCOD_Major_Mask.
btLibCOD_Minor_Any
Used as a device filter for the
BtLibDiscoverMultipleDevices and
BtLibDiscoverSingleDevice functions.
With this filter, devices in any minor device
class appear in the device list. Same as
btLibCOD_Minor_Mask.

btLibCOD_Minor_Comp_Any
Used as a device filter for the
BtLibDiscoverMultipleDevices and
BtLibDiscoverSingleDevice functions.
When this filter is used in conjunction with
btLibCOD_Major_Computer, all devices
broadcasting themselves as computers appear
in the device list. Same as
btLibCOD_Minor_Any.

btLibCOD_Minor_Phone_Any
Used as a device filter for the
BtLibDiscoverMultipleDevices and
BtLibDiscoverSingleDevice functions.
When this filter is used in conjunction with
btLibCOD_Major_Phone, all devices
broadcasting themselves as phones appear in
the device list. Same as
btLibCOD_Minor_Any.

btLibCOD_Minor_LAN_Any
Used as a device filter for the
BtLibDiscoverMultipleDevices and
BtLibDiscoverSingleDevice functions.
When this filter is used in conjunction with
btLibCOD_Major_Lan_Access_Point, all
devices broadcasting themselves as LAN access
points appear in the device list. Same as
btLibCOD_Minor_Any.
btLibCOD_Minor_Audio_Any
Used as a device filter for the
BtLibDiscoverMultipleDevices and
BtLibDiscoverSingleDevice functions.
When this filter is used in conjunction with
btLibCOD_Major_Audio, all devices
broadcasting themselves as audio devices
appear in the device list. Same as
btLibCOD_Minor_Any.

New
BtLibConnectionRoleEnum
The BtLibConnectionRoleEnum enum specifies all the
connection roles a device can have. A device can either be a master
or a slave.

typedef enum {
    btLibMasterRole,
    btLibSlaveRole
} BtLibConnectionRoleEnum;

Value Descriptions
btLibMasterRole    The device is a master.
btLibSlaveRole     The device is a slave.

New
BtLibDeviceAddressType
The BtLibDeviceAddressType structure represents a 48-bit
Bluetooth device address.

#define btLibDeviceAddressSize 6

typedef struct BtLibDeviceAddressType {
    UInt8 address[btLibDeviceAddressSize];
} BtLibDeviceAddressType;
New BtLibFriendlyNameType

The BtLibFriendlyNameType structure contains the user-friendly name of a device.

typedef struct BtLibFriendlyNameType {
    UInt8  *name;
    UInt8  nameLength;
    BtLibFriendlyNameType,
    *BtLibFriendlyNameTypePtr;
} BtLibFriendlyNameType;

Field Descriptions

name Array of characters, encoded according to the UTF-8 standard, containing the user-friendly name of the device. This array is not null-terminated.

nameLength The number of characters in the user-friendly name. The maximum size is 249 characters.

New BtLibManagementEventType

The BtLibManagementEventType structure contains detailed information regarding a management callback event. All management events have some common data. Most management events have data specific to those events. The specific data uses a union that is part of the BtLibManagementEvent data structure.

typedef struct _BtLibManagementEventType {
    BtLibManagementEventEnum event;
    Err status;
    union {
        ...
    } eventData;
} BtLibManagementEventType;
Field Descriptions

**event**
A `BtLibManagementEventEnum` enum member that indicates which management event has occurred. See [Management Callback Events](#).

**status**
Status of the event. The [Management Callback Events](#) section gives more details about how to interpret this field for specific events.

**eventData**
Data associated with the event. The member of this union that is valid depends on the event. See [Management Callback Events](#) for more information.

A `BtLibManagementEventType` object is passed as the first argument of the `BtLibManagementCallback` callback function.

Management Callback Events

The management functions of the Bluetooth library support the lower levels of the Bluetooth specification, specifically the radio, baseband, and Link Manager Protocol specifications. Most of the management functions are asynchronous. In other words, they start operations and return before the operations complete. To signal the application that management operations have completed, the Bluetooth library generates *management callback events* by calling a callback function.

You specify the callback function using the `BtLibRegisterManagementNotification` function. When an event occurs, the callback function is called with two parameters: a pointer to a `BtLibManagementEventType` structure and a pointer to a user-defined structure.

The `BtLibManagementEventType` structure contains an event field, which indicates the reason the callback is called, a status field, which contains status information associated with the event, and a union of several structures. The member of the union that is valid depends on the event. The meaning of the events is described in the following sections.
For more information about the status field, see Management Event Status Codes.

**btLibManagementEventAccessibilityChange**

The accessibility mode of the local device has changed.

For this event, the eventData field contains the following field:

```c
BtLibAccessibleModeEnum accessible;
```

This `BtLibAccessibleModeEnum` represents the new accessibility mode of the local device.

This event can result from calling `BtLibOpen`, `BtLibPiconetCreate`, or `BtLibSetGeneralPreference`.

**btLibManagementEventAclConnectInbound**

A remote device has established an ACL link to the local device.

For this event, the eventData field contains the following field:

```c
BtLibDeviceAddressType bdAddr;
```

This `BtLibDeviceAddressType` contains the address of the remote device.

**btLibManagementEventAclConnectOutbound**

An ACL link has been established with a remote device.

If the status field contains `btLibErrNoError`, the ACL link is connected, and the remote device address can be found in the eventData field. Otherwise the connection failed and the status field indicates the reason for the failure. See Management Event Status Codes for more information.

For this event, the eventData field contains the following field:

```c
BtLibDeviceAddressType bdAddr;
```

This `BtLibDeviceAddressType` contains the address of the remote device.

This event can result from calling `BtLibLinkConnect`.
Bluetooth Library: Management

Management Callback Events

btLibManagementEventAclDisconnect
An ACL link has been disconnected. The status field indicates the reason the link was disconnected. See Management Event Status Codes.

For this event, the eventData field contains the following field:

```c
    BtLibDeviceAddressType bdAddr;
```

This `BtLibDeviceAddressType` contains the address of the disconnected device.

This event can result from calling `BtLibLinkDisconnect` or `BtLibPiconetDestroy`.

btLibManagementEventAuthenticationComplete
The authentication of a remote device has completed.

For this event, the eventData field contains the following field:

```c
    BtLibDeviceAddressType bdAddr;
```

This `BtLibDeviceAddressType` contains the address of the remote device.

If the authentication is successful, the status field contains `btLibErrNoError`. If the user cancels the passkey request, the status field contains `btLibErrCanceled`. Otherwise, the status field indicates the reason the authentication failed. See Management Event Status Codes.

This event can result from calling `BtLibLinkSetState`.

btLibManagementEventEncryptionChange
Encryption for a link has been enabled or disabled.

For this event, the eventData field contains the following structure:

```c
    struct {
        BtLibDeviceAddressType bdAddr;
        Boolean enabled;
    } encryptionChange;
```
Field Descriptions

bdAddr    A BtLibDeviceAddressType containing the address of the remote device.

enabled  true when encryption for the link has been enabled; false otherwise.

This event can result from calling BtLibLinkSetState.

btLibManagementEventInquiryCanceled

The device inquiry has been canceled because the application called BtLibCancelInquiry.

btLibManagementEventInquiryComplete

The device inquiry started with the BtLibStartInquiry function has completed.

btLibManagementEventInquiryResult

A remote device has responded to an inquiry that was started with the BtLibStartInquiry function.

For this event, the eventData field contains the following structure:

struct {
    BtLibDeviceAddressType bdAddr;
    BtLibClassOfDeviceType classOfDevice;
} inquiryResult;

Field Descriptions

bdAddr    A BtLibDeviceAddressType containing the address of the remote device.

classOfDevice  A BtLibClassOfDeviceType representing the class of the remote device.

btLibManagementEventLocalNameChange

The user-friendly name of the local device has changed.
For this event, the eventData field contains the following structure:

```c
struct {
    BtLibDeviceAddressType bdAddr;
    BtLibFriendlyNameType name;
} nameResult;
```

**Field Descriptions**

- **bdAddr**  
  A `BtLibDeviceAddressType` containing the address of the local device.

- **name**  
  A `BtLibFriendlyNameType` containing the new name.

This event can result from calling `BtLibOpen`.

### `btLibManagementEventModeChange`

A slave has changed its mode. A slave can be in active, sniff, hold, or park mode.

For this event, the eventData field contains the following structure:

```c
struct {
    BtLibDeviceAddressType bdAddr;
    BtLibLinkModeEnum curMode;
    UInt16 interval;
} modeChange;
```

**Field Descriptions**

- **bdAddr**  
  A `BtLibDeviceAddressType` containing the address of the remote device.

- **curMode**  
  A `BtLibLinkModeEnum` indicating the new mode of remote device

- **interval**  
  The time in units of 0.625 ms the remote device will stay in the new mode, if applicable. The time period is a standard time period in the Bluetooth specification.
BtLibLinkModeEnum

The BtLibLinkModeEnum enum specifies the modes a slave can have. According to the *Specification of the Bluetooth System*, a slave can be in active, sniff, hold, or park mode. However, the Bluetooth library only supports the hold and active modes.

```c
typedef enum {
    btLibSniffMode,
    btLibHoldMode,
    btLibParkMode,
    btLibActiveMode
} BtLibLinkModeEnum;
```

**Value Descriptions**

- **btLibActiveMode**: The slave is active.
- **btLibHoldMode**: The slave is in hold mode.
- **btLibParkMode**: The slave is in park mode. This mode is not currently supported.
- **btLibSniffMode**: The slave is in sniff mode. This mode is not currently supported.

**btLibManagementEventNameResult**

A remote device name request has completed. If the `status` field is `btLibErrNoError`, the name is available. Otherwise, the name request failed, and the `status` field indicates the reason for the failure. See *Management Event Status Codes*.

For this event, the `eventData` field contains the following structure:

```c
struct {
    BtLibDeviceAddressType bdAddr;
    BtLibFriendlyNameType name;
} nameResult;
```

**Field Descriptions**

- **bdAddr**: A `BtLibDeviceAddressType` containing the address of the remote device.
Bluetooth Library: Management
Management Callback Events

name A BtLibFriendlyNameType containing the name of the remote device.

The BtLibGetRemoteDeviceName function is used to start a remote device name request.

btLibManagementEventPasskeyRequest
A remote device has requested a passkey. Your application does not have to respond to this request—the Bluetooth library automatically handles it.

For this event, the eventData field contains the following field:

   BtLibDeviceAddressType bdAddr;

This BtLibDeviceAddressType contains the address of the remote device.

Because a passkey can be requested during or after a link is established, consider disabling any failure timers while the passkey dialog is up. The btLibManagementEventPasskeyRequestComplete event signals the completion of the passkey entry.

btLibManagementEventPasskeyRequestComplete
A passkey request has been processed. The status code for this event is set to btLibErrNoError if the passkey was entered or btLibErrCanceled if passkey entry was cancelled. Note that this event does not tell you that the authentication completed.

btLibManagementEventPiconetCreated
The piconet has been created. This event can result from calling BtLibPiconetCreate.

btLibManagementEventPiconetDestroyed
The piconet has been destroyed. This event can result from calling BtLibPiconetDestroy.
btLibManagementEventRadioState

This event is generated when the Bluetooth radio changes state. The radio changes state when the radio is disconnected, the power is turned on or off, the radio resets, or the radio fails to initialize. The status code for this event explains why the event gets generated.

Status Codes

btLibErrRadioInitialized

The Bluetooth radio has initialized successfully. You can now call management functions.

btLibErrRadioInitFailed

The Bluetooth radio failed to initialize. The application can assume all pending Bluetooth operations have failed. However, some pending operations will still generate events and modify memory supplied by the application.

To try to initialize the radio again, you need to close the library and reopen it.
**btLibErrRadioFatal**

A fatal radio error occurred. This usually signifies that the host has lost contact with the radio, for example, when the user disconnects the radio, or the device turns off. The application can assume that all pending operations have failed. However, some pending operations will still generate events and modify memory supplied by the application.

When a fatal radio error occurs, the Bluetooth stack resets the radio and tries once to reinitialize it, which generates another btLibManagementEventRadioState event with a status code of btLibErrRadioInitialized, or btLibErrRadioInitFailed depending on whether or not the initialization succeeded.

**btLibErrRadioSleepWake**

The radio was reset because the device went to sleep. The application can assume all pending operations have failed. However, some pending operations will still generate events and modify memory supplied by the application.

The Bluetooth stack resets the radio and tries once to reinitialize it, which generates another btLibManagementEventRadioState event with a status code of btLibErrRadioInitialized, or btLibErrRadioInitFailed depending on whether or not the initialization succeeded.

This event can result from calling **BtLibOpen**.

**btLibManagementEventRoleChange**

The master and slave devices for a link have switched roles.
For this event, the `eventData` field contains the following structure:

```c
struct {
    BtLibDeviceAddressType bdAddr;
    BtLibConnectionRoleEnum newRole;
} roleChange;
```

**Field Descriptions**

- **bdAddr** A `BtLibDeviceAddressType` containing the address of the remote device.
- **newRole** A `BtLibConnectionRoleEnum` representing the new role of the local device.

## Management Event Status Codes

When a management event is generated, the `status` field of the associated `BtLibManagementEventType` provides information about why the event occurred. The following status codes can occur with a management event.

- **btLibErrNoError** Success.
- **btLibMeStatusAuthenticateFailure** Authentication failure.
- **btLibMeStatusCommandDisallowed** Command disallowed.
- **btLibMeStatusConnectionTimeout** Connection timeout.
- **btLibMeStatusHardwareFailure** Hardware Failure.
- **btLibMeStatusHostTimeout** Host timeout.
- **btLibMeStatusInvalidHciParam** Invalid HCI command parameters.
- **btLibMeStatusInvalidLmpParam** Invalid LMP Parameters.
btLibMeStatusLimitedResources
    Host rejected due to limited resources
btLibMeStatusLmpResponseTimeout
btLibMeStatusLmpTransdCollision
btLibMeStatusLmpPduNotAllowed
btLibMeStatusLocalTerminated
    Terminated by local host
btLibMeStatusLowResources
    Other end terminated due to low resources
btLibMeStatusMaxAclConnections
    Max number of ACL connections to a device
btLibMeStatusMaxConnections
    Max number of connections
btLibMeStatusMaxScoConnections
    Max number of SCO connections to a device
btLibMeStatusMemoryFull
    Memory full
btLibMeStatusMissingKey
    Missing key
btLibMeStatusNoConnection
    No connection
btLibMeStatusPageTimeout
    Page timeout
btLibMeStatusPairingNotAllowed
    Pairing not allowed
btLibMeStatusPersonalDevice
    Host rejected (remote is personal device)
bLibMeStatusPowerOff
    Other end terminated (about to power off)
btLibMeStatusRepeatedAttempts
Repeated attempts

btLibMeStatusRoleChangeNotAllowed
Change not allowed

btLibMeStatusScoAirModeRejected
SCO Air Mode Rejected

btLibMeStatusScoIntervalRejected
SCO Interval Rejected

btLibMeStatusScoOffsetRejected
SCO Offset Rejected

btLibMeStatusSecurityError
Host rejected due to security reasons

btLibMeStatusUnknownHciCommand
Unknown HCI Command

btLibMeStatusUnknownLmpPDU
Unknown LMP PDU

btLibMeStatusUnspecifiedError
Unspecified Error

btLibMeStatusUnsupportedFeature
Unsupported feature or parameter value

btLibMeStatusUnsupportedLmpParam
Unsupported LMP Parameter Value

btLibMeStatusUnsupportedRemote
Unsupported Remote Feature

btLibMeStatusUserTerminated
Other end terminated (user)

Library Management Functions
This section describes the general Bluetooth library management functions.
### New

**BtLibClose**

**Purpose**
Close the Bluetooth library.

**Declared In**
BtLib.h

**Prototype**
Err BtLibClose (UInt16 btLibRefNum)

**Parameters**
- `btLibRefNum` Reference number for the Bluetooth library.

**Result**
Returns `btLibErrNoError` if the library successfully closes. Returns `btLibErrNotOpen` if the referenced Bluetooth library was not open.

**Comments**
Applications must call this function when they no longer need the Bluetooth library. If the Bluetooth library open count is one, this function closes existing connections, saves the current accessibility mode, sets the accessible mode according to the preferences panel, and shuts down the library. If the Bluetooth library open count is greater than one, this function decrements the open count.

**See Also**
BtLibOpen

---

### New

**BtLibOpen**

**Purpose**
Open and initialize the Bluetooth library.

**Declared In**
BtLib.h

**Prototype**
Err BtLibOpen (UInt16 btLibRefNum, Boolean allowStackToFail)

**Parameters**
- `btLibRefNum` Reference number for the Bluetooth library.
Bluetooth Library: Management
Library Management Functions

-> allowStackToFail
If true, opens the library even if the stack or radio fails to initialize. Otherwise, does not open the library if the stack or radio fails to initialize.

Result
Returns one of the following values:

btLibErrNoError
Success.

btLibErrAlreadyOpen
This status code is not really an error. It is returned if the library was already open. In this case, the open count is incremented.

btLibErrInUseByService
The library is currently in use by a Bluetooth service.

btLibErrOutOfMemory
Not enough memory available to open the library.

btLibErrRadioInitFailed
The Bluetooth stack or radio could not be initialized. If allowStackToFail is true, the library still opens after this error occurs.

Applications must call this function before using the Bluetooth library. If the Bluetooth library is not already open, BtLibOpen opens the library, initializes it, and starts up the protocol stack component of the library. Otherwise it increments its open count.

The allowStackToFail parameter allows the library to be opened even if the Bluetooth stack or radio fails to initialize. It is useful for applications that only want to use the Bluetooth library’s utility functions but not the radio. However, any application that needs to communicate with the radio must set allowStackToFail to false.

This function generates three events: a btLibManagementEventRadioState event with a status of btLibErrRadioInitialized, a btLibManagementEventLocalNameChange event indicating the
local name of the device, and a
btLibManagementEventAccessibilityChange event
indicating the accessibility of the device.

See Also  BtLibClose

Management Functions

The management functions handle the lower levels of the Bluetooth
specification, specifically the radio, baseband, and Link Manager
Protocol specifications. These functions perform tasks that include
discovering devices, working with Asynchronous Connectionless
(ACL) links and piconets, and maintaining global settings for the
Bluetooth library.

New
BtLibCancelInquiry

Purpose  Cancel a Bluetooth inquiry in process.

Declared In  BtLib.h

Prototype  Err BtLibCancelInquiry (UInt16 btLibRefNum)

Parameters  -> btLibRefNum  Reference number for the Bluetooth library.

Result  Returns one of the following values:

btLibErrNoError  The inquiry process was canceled before it started.

btLibErrPending  The cancellation is pending. When it succeeds,
notification will be provided through a
management callback event.

btLibErrInProgress  The inquiry is already being canceled.
btLibErrNotInProgress
   No inquiry is in progress to be canceled.

btLibErrNotOpen
   The referenced Bluetooth library is not open.

btLibErrStackNotOpen
   The Bluetooth stack failed to initialize when the library was opened.

Comments
The function cancels inquiries initiated by BtLibStartInquiry. The btLibManagementEventInquiryCanceled callback event indicates that the cancellation has completed.

A Bluetooth discovery initiated using either BtLibDiscoverSingleDevice or BtLibDiscoverMultipleDevices cannot be canceled with this function. Only the user can cancel these inquiries by tapping the Cancel button.

See Also  BtLibStartInquiry

New  BtLibDiscoverMultipleDevices

Purpose  Discover all available devices, present them in the user interface, and allow the user to select one or more of these devices.

Declared In  BtLib.h

Prototype
Err  BtLibDiscoverMultipleDevices
   (UInt16  btLibRefNum, Char  *instructionTxt,
    Char  *buttonTxt,
    BtLibClassOfDeviceType  *deviceFilterList,
    UInt8  deviceFilterListLen,
    UInt8  *numDevicesSelected, Boolean  addressAsName,
    Boolean  showLastList)

Parameters
   ->  btLibRefNum  Reference number for the Bluetooth library.
Bluetooth Library: Management
Management Functions

-> instructionTxt
   Text displayed at the top of the selection box. Pass NULL to display the default text. The default text is “Select one or more devices:”

-> buttonTxt
   Text for the OK button. Pass NULL to display the default text. The default button text is “OK”

-> deviceFilterList
   Array of BtLibClassOfDeviceTypes. This function presents to the user only the remote devices whose class matches a class in this list. If deviceFilterList is NULL, this function presents to the user all discovered devices.

-> deviceFilterListLen
   Number of elements in deviceFilterList.

<- numDevicesSelected
   Number of selected devices. To obtain the actual device list, use the BtLibGetSelectedDevices function.

-> addressAsName
   If true, display the Bluetooth addresses of the remote devices instead of their names. This option is available for debugging purposes.

-> showLastList
   If true, causes all other parameters to be ignored and displays the same list as the previous call to BtLibDiscoverMultipleDevices.

Result
   Returns one of the following values:

   btLibErrNoError
      Success

   btLibErrCanceled
      User canceled discovery.

   btLibErrNotOpen
      The referenced Bluetooth library is not open.
btLibErrStackNotOpen
The Bluetooth stack failed to initialize when the library was opened.

Comments
This blocking call performs a full discovery for an application, including name and feature retrieval and testing. This function takes over the UI and presents a choice box to the user, allowing the user to select multiple devices from the list of devices that were discovered. This function does not return until the user chooses one or more devices, or cancels.

Setting the showLastList parameter to true allows you to present to the user the list of devices displayed in the previous call to BtLibDiscoverMultipleDevices or BtLibDiscoverSingleDevice. This feature can be useful because a full discovery process takes approximately ten seconds. The cached device list remains valid even after you close the library, allowing other Bluetooth applications to use it.

Note that BtLibStartInquiry overwrites the cached device list. If you are using the showLastList feature, you should avoid calling BtLibStartInquiry between calls to BtLibDiscoverMultipleDevices or BtLibDiscoverSingleDevice.

Use BtLibGetSelectedDevices to retrieve the list of devices that the user selected.

See Also
BtLibGetSelectedDevices, BtLibDiscoverSingleDevice
New  **BtLibDiscoverSingleDevice**

**Purpose**
Discover all available devices, present them in the user interface, and allow the user to select one of these devices.

**Declared In**
BtLib.h

**Prototype**
```c
Err BtLibDiscoverSingleDevice
(UInt16 btLibRefNum, Char *instructionTxt,
BtLibClassOfDeviceType *deviceFilterList,
UInt8 deviceFilterListLen,
BtLibDeviceAddressType *selectedDeviceP,
Boolean addressAsName, Boolean showLastList)
```

**Parameters**
- `-> btLibRefNum` Reference number for the Bluetooth library.
- `-> instructionTxt` Text displayed at the top of the selection box. Pass NULL to display the default text. The default text is “Select a device:”
- `-> deviceFilterList` Array of **BtLibClassOfDeviceTypes**. This function displays only the remote devices whose class matches a class in this list. If `deviceFilterList` is NULL, this function displays all discovered devices.
- `-> deviceFilterListLen` Number of elements in `deviceFilterList`.
- `<- selectedDeviceP` Pointer to a **BtLibDeviceAddressType** where this function stores the address of the device the user selects. You need to allocate this space before calling this function.
- `-> addressAsName` If true, display the Bluetooth addresses of the remote devices instead of their names. This option is available for debugging purposes.
-> showLastList
  If true, causes all other parameters to be ignored and displays the same list as the previous call to BtLibDiscoverSingleDevice.

Result
Returns one of the following values:

- btLibErrNoError
  Success
- btLibErrCanceled
  User canceled the discovery.
- btLibErrNotOpen
  The referenced Bluetooth library is not open.
- btLibErrStackNotOpen
  The Bluetooth stack failed to initialize when the library was opened.

Comments
This blocking call performs a full discovery for an application, including name and feature retrieval and testing. This function takes over the UI and presents a choice box to the user, allowing the user to select a device from the list of devices that were discovered. This function does not return until the user chooses a device or cancels.

Setting the showLastList parameter to true allows you to present to the user the list of devices displayed in the previous call to BtLibDiscoverSingleDevice or BtLibDiscoverMultipleDevices. This feature can be useful because a full discovery process takes approximately ten seconds. The cached device list remains valid even after you close the library, allowing other Bluetooth applications to use it.

Note that BtLibStartInquiry overwrites the cached device list. If you are using the showLastList feature, you should avoid calling BtLibStartInquiry between calls to BtLibDiscoverSingleDevice or BtLibDiscoverMultipleDevices.

See Also
BtLibDiscoverMultipleDevices
New **BtLibGetGeneralPreference**

**Purpose**
Get one of the general management preferences.

**Declared In**
BtLib.h

**Prototype**
```
Err BtLibGetGeneralPreference
(UInt16 btLibRefNum, BtLibGeneralPrefEnum pref,
void *prefValue, UInt16 prefValueSize)
```

**Parameters**
- `btLibRefNum` Reference number for the Bluetooth library.
- `pref` General preference to get.
- `prefValue` Pointer to a buffer to hold the value of the preference. You must allocate this buffer. This parameter must not be NULL.
- `prefValueSize` Size, in bytes, of the `prefValue` buffer. You must set this size so it matches the size of the retrieved preference.

**Result**
Returns one of the following values:
- `btLibErrNoError` Success.
- `btLibErrNotOpen` The referenced Bluetooth library is not open.
- `btLibErrParamError` One or more parameters is invalid. Be sure that the `prefValueSize` parameter matches the size of the preference value.
- `btLibErrStackNotOpen` The Bluetooth stack failed to initialize when the library was opened.
Comments
Specify the preference with a member of the BtLibGeneralPreferenceEnum.

**BtLibGeneralPreferenceEnum**

The BtLibGeneralPreferenceEnum enum specifies the general preferences that can be accessed using the BtLibSetGeneralPreference and BtLibGetGeneralPreference functions.

```c
typedef enum {
    btLibPref_Name,
    btLibPref_UnconnectedAccessible,
    btLibPref_CurrentAccessible,
    btLibPref_LocalClassOfDevice,
    btLibPref_LocalDeviceAddress
} BtLibGeneralPrefEnum;
```

**Value Descriptions**

**btLibPref_CurrentAccessible**

This preference is a BtLibAccessibleModeEnum indicating the current accessibility mode of the local device.

**btLibPref_LocalClassOfDevice**

This preference is a BtLibClassOfType indicating the class of the local device. You should never set this preference.

**btLibPref_Name**

This preference is a BtLibFriendlyNameType containing the user-friendly name of the local device. If you retrieve this preference, you also need to allocate a buffer and set the BtLibFriendlyNameType’s name and nameLength fields to the buffer pointer and buffer length, respectively.

You should never set this preference.
btLibPref_UnconnectedAccessible
This preference is a BtLibAccessibleModeEnum indicating the accessibility mode of the local device when it is unconnected. You should never set this preference.

See Also BtLibSetGeneralPreference

New BtLibGetRemoteDeviceName

Purpose Get the name of the remote device with the specified address.

Declared In BtLib.h

Prototype Err BtLibGetRemoteDeviceName (UInt16 btLibRefNum, BtLibDeviceAddressTypePtr remoteDeviceP, BtLibFriendlyNameType *nameP, BtLibGetNameEnum retrievalMethod)

Parameters -> btLibRefNum Reference number for the Bluetooth library.

-> remoteDeviceP Pointer to a BtLibDeviceAddressType containing the address of the device whose name is desired.

<-> nameP Pointer to a BtLibFriendlyNameType structure in which to store the results of the lookup. You must allocate this structure and the name buffer it points to. You also must specify the size of the buffer in the nameLength field of the structure. When the function returns, the nameLength field contains the actual length of the name. This parameter must not be NULL.
-> retrievalMethod
    Method used to retrieve the user-friendly remote device name. See BtLibGetNameEnum.

**Result**

Returns one of the following values:

- btLibErrNoError
  The name structure was successfully retrieved from the cache. No notification event will be generated.

- btLibErrBusy
  There is already a name request pending.

- btLibErrNotOpen
  The referenced Bluetooth library is not open.

- btLibErrPending
  The results will be returned through a notification.

- btLibErrStackNotOpen
  The Bluetooth stack failed to initialize when the library was opened.

**Comments**

The Bluetooth library maintains a cache of 50 device names. If the retrievalMethod parameter is btLibCachedThenRemote, this function first checks the cache for a name. If the name is in the cache, the value is returned immediately in the nameP parameter. If the name is not in the cache, the function queries the remote device for its name, forming a temporary ACL connection if one is not already in place. In this case, the function returns btLibErrPending and generates a BtLibManagementEventNameResult event when the name is available.

Other values of the retrievalMethod parameter can instruct this function to look for the name only in the cache or only on the remote device. See BtLibGetNameEnum for more information.

**BtLibGetNameEnum**

The BtLibGetNameEnum enum specifies whether to retrieve a device name from the cache, the remote device, or both.
typedef enum {
    btLibCachedThenRemote,
    btLibCachedOnly,
    btLibRemoteOnly
} BtLibGetNameEnum;

**Value Descriptions**

- **btLibCachedOnly**
  Look for a name in the cache. If the name is not in the cache, fail.

- **btLibCachedThenRemote**
  Look for a name in the cache. If the name is not in the cache, ask the remote device.

- **btLibRemoteOnly**
  Ignore any cached names and ask the remote device for its name.

---

**New**

**BtLibGetSelectedDevices**

**Purpose**
Get the list of devices selected during the last call to `BtLibDiscoverMultipleDevices`.

**Declared In**
`BtLib.h`

**Prototype**
```
Err BtLibGetSelectedDevices (UInt16 btLibRefNum,
                            BtLibDeviceAddressType *selectedDeviceArray,
                            UInt8 arraySize, UInt8 *numDevicesReturned)
```

**Parameters**

- `-> btLibRefNum` Reference number for the Bluetooth library.
- `<->selectedDeviceArray` Array into which the results of the `BtLibDiscoverMultipleDevices` function should be placed. You must allocate this array of `BtLibDeviceAddressTypes`.
- `-> arraySize` Number of elements in the `selectedDeviceArray` you allocated.
<numDevicesReturned>

Number of results placed in selectedDeviceArray.

**Result**

Returns btLibErrNoError if the query is successful. Returns btLibErrNotOpen if the referenced Bluetooth library is not open or btLibErrStackNotOpen if the Bluetooth stack failed to initialize when the library was opened.

**Comments**

No callback events.

**See Also**

BtLibDiscoverMultipleDevices

---

**New**

**BtLibLinkConnect**

**Purpose**

Create a Bluetooth Asynchronous Connectionless (ACL) link.

**Declared In**

BtLib.h

**Prototype**

Err BtLibLinkConnect (UInt16 btLibRefNum, BtLibDeviceAddressTypePtr remoteDeviceP)

**Parameters**

- `-> btLibRefNum` Reference number for the Bluetooth library.
- `-> remoteDeviceP` Pointer to the a BtLibDeviceAddressType containing the address of the remote device.

**Result**

Returns one of the following values:

- `btLibErrPending` The results will be returned through a callback event.
- `btLibErrAlreadyConnected` The device is already in a pre-existing connection and cannot create a new connection.
Bluetooth Library: Management
Management Functions

btLibErrBluetoothOff
The Bluetooth radio is off. The user can turn the radio on and off with a setting in the preferences panel.

btLibErrBusy
A piconet is currently being created or destroyed.

btlibErrNotOpen
The referenced Bluetooth library is not open.

btLibErrStackNotOpen
The Bluetooth stack failed to initialize when the library was opened.

btLibErrTooMany
Cannot create another ACL link because the maximum allowed number has already been reached.

Comments
An ACL link is a packet-switched physical level connection between two devices that is needed before the devices can form a RFCOMM or L2CAP connection.

When the connection is established or if it fails to be established, the btLibManagementEventAclConnectOutbound event is generated.

See Also
BtLibLinkDisconnect

New
BtLibLinkDisconnect

Purpose
Disconnect an existing ACL Link.

Declared In
BtLib.h

Prototype
Err BtLibLinkDisconnect (UInt16 btLibRefNum,
BtLibDeviceAddressTypePtr remoteDeviceP)

Parameters
- > btLibRefNum  Reference number for the Bluetooth library.
Pointer to a BtLibDeviceAddressType containing the address of the remote device.

Returns one of the following values:

- **btLibErrNoError**: The connection attempt was canceled before it started. No event is generated.
- **btLibErrPending**: When the link actually disconnects, a btLibManagementEventAclDisconnect callback event is generated.
- **btLibErrBusy**: Can’t disconnect the link because the piconet is being destroyed.
- **btLibErrNoConnection**: No link to the specified device exists.
- **btLibErrNotOpen**: The referenced Bluetooth library is not open.
- **btLibErrStackNotOpen**: The Bluetooth stack failed to initialize when the library was opened.

When the link disconnects, a btLibManagementEventAclDisconnect event is generated.

See Also: BtLibLinkConnect
New **BtLibLinkGetState**

**Purpose**
Get the state of an ACL link.

**Declared In**
BtLib.h

**Prototype**
```c
Err BtLibLinkGetState(UInt16 btLibRefNum,
                       BtLibDeviceAddressTypePtr remoteDeviceP,
                       BtLibLinkPrefsEnum pref, void *linkState,
                       UInt16 linkStateSize)
```

**Parameters**
- `btLibRefNum` Reference number for the Bluetooth library.
- `remoteDeviceP` Pointer to a `BtLibDeviceAddressType` containing the address of the remote device. This address identifies the ACL link.
- `pref` Link preference to retrieve. See `BtLibLinkPrefsEnum`.
- `linkState` Pointer to a buffer to store the value of the preference. You must allocate this buffer. This parameter must not be NULL. See `BtLibLinkPrefsEnum` for more information.
- `linkStateSize` Size, in bytes, of `linkState` buffer. This size must match the size of the retrieved preference.

**Result**
Returns one of the following values:
- `btLibErrNoError` Success. The `linkState` variable has been filled in.
- `btLibErrNoAclLink` No link to the specified remote device exists.
- `btLibErrNotOpen` The referenced Bluetooth library is not open.
btLibErrParamError
The linkStateSize parameter is not same as the size of the preference value.

btLibErrStackNotOpen
The Bluetooth stack failed to initialize when the library was opened.

Comments
See the BtLibLinkSetState function description for a list of the link states preferences.

See Also
BtLibLinkSetState

New
BtLibLinkSetState

Purpose
Set the state of an ACL link

Declared In
BtLib.h

Prototype
Err BtLibLinkSetState (UInt16 btLibRefNum, BtLibDeviceAddressTypePtr remoteDeviceP, BtLibLinkPrefsEnum pref, void *linkState, UInt16 linkStateSize)

Parameters
-> btLibRefNum  Reference number for the Bluetooth library.
-> remoteDeviceP  The address of the remote device. This address identifies the ACL link.
-> pref  Link preference to set. See BtLibLinkPrefsEnum.
-> linkState  Pointer to the value of the preference. This parameter must not be NULL. See BtLibLinkPrefsEnum.
Bluetooth Library: Management
Management Functions

-> linkStateSize
  Size, in bytes, of the linkState value.

Result
Returns one of the following values:

btLibErrPending
  The results will be returned through a callback event.

btLibErrFailed
  An attempt was made to encrypt a link before authenticating it.

btLibErrNoAclLink
  No link to the specified remote device exists.

btLibErrNotOpen
  The referenced Bluetooth library is not open.

btLibErrParamError
  The preference cannot be set or linkStateSize is invalid.

btLibErrStackNotOpen
  The Bluetooth stack failed to initialize when the library was opened.

Comments
Applications use this function to set the state of an ACL link. This function may generate events depending on the preference you change. The btLibManagementEventAuthentication Complete event indicates the link authentication has completed. The btLibManagementEventEncryptionChange indicates that the encryption has been enabled or disabled.

BtLibLinkPrefsEnum
The BtLibLinkPrefsEnum enum specifies the link state preferences that can be accessed with the BtLibLinkSetState and BtLibLinkGetState functions.

```c
typedef enum {
    btLibLinkPref_AUTHENTICATED,
    btLibLinkPref_ENCRYPTED,
    btLibLinkPref_LINKROLE
} BtLibLinkPrefsEnum;
```
**Value Descriptions**

btLibLinkPref_Authenticated
---
This preference is a Boolean and indicates whether the link has been authenticated or not.

btLibLinkPref_Encrypted
---
This preference is a Boolean and indicates whether the link is encrypted or not.

btLibLinkPref_LinkRole
---
This preference is a BtLibConnectionRoleEnum and indicates whether the remote device is a master or a slave. You cannot set this preference but you can get its value.

**See Also**

BtLibLinkGetState

---

**New**

BtLibPiconetCreate

**Purpose**
Create a piconet or reconfigure an existing piconet so the local device is the master.

**Declared In**
BtLib.h

**Prototype**
Err BtLibPiconetCreate (UInt16 btLibRefNum, Boolean unlockInbound, Boolean discoverable)

**Parameters**
- `btLibRefNum` Reference number for the Bluetooth library.
- `unlockInbound` If true, the piconet accepts inbound connections. Otherwise, the piconet only allows outbound connections.
Bluetooth Library: Management
Management Functions

-> discoverable

If true, configures the radio to be discoverable. In other words, the radio responds to inquiries. If false, configures the radio to be only connectable. In other words, only devices that know the radio’s Bluetooth device address can connect to it. This parameter is ignored if unlockInbound is false.

Result

Returns one of the following values:

btLibErrNoError
Successfully created the piconet with the local device as the master. No callback event is generated.

btLibErrPending
An existing ACL link needs to switch roles before this operation can complete.

btLibErrFailed
A piconet already exists.

btLibErrInProgress
Another piconet is currently being created.

btLibErrNotOpen
The referenced Bluetooth library is not open.

btLibErrStackNotOpen
The Bluetooth stack failed to initialize when the library was opened.

Comments

If the accessibility of the radio changes due to this operation, a btLibManagementEventAccessibilityChange event is generated. When the piconet is created, or if the piconet fails to be created, a btLibManagementEventPiconetCreated event is generated. The status field of the BtLibManagementEventType structure accompanying the event indicates whether the piconet was created or not.

See Also

BtLibPiconetDestroy, BtLibPiconetUnlockInbound, BtLibPiconetLockInbound
**BtLibPiconetDestroy**

**Purpose**
Destroy the piconet by disconnecting links to all devices and removing all restrictions on whether the local device is a master or a slave.

**Declared In**
BtLib.h

**Prototype**
Err BtLibPiconetDestroy (UInt16 btLibRefNum)

**Parameters**
-> btLibRefNum  Reference number for the Bluetooth library.

**Result**
Returns one of the following values:

- **btLibErrNoError**
  Successfully destroyed the piconet. A **btLibManagementEventPiconetDestroyed** event is not generated.

- **btLibErrPending**
  The piconet is being destroyed, and a **btLibManagementEventPiconetDestroyed** event will be generated when the operation succeeds or fails.

- **btLibErrBusy**
  The piconet is already in the process of being destroyed.

- **btLibErrNoPiconet**
  No piconet exists to be destroyed.

- **btLibErrNotOpen**
  The referenced Bluetooth library is not open.

- **btLibErrStackNotOpen**
  The Bluetooth stack failed to initialize when the library was opened.

**Comments**
A **btLibManagementEventAclDisconnect** event is generated for each ACL link that is disconnected. When the piconet is successfully destroyed or fails to be destroyed, a
btLibManagementEventPiconetDestroyed is generated. The status field of the BtLibManagementEventType structure accompanying the event indicates whether the piconet was destroyed or not.

See Also  BtLibPiconetCreate

New  BtLibPiconetLockInbound

Purpose  Prevent remote devices from creating ACL links into the piconet.

Declared In  BtLib.h

Prototype  Err BtLibPiconetLockInbound (UInt16 btLibRefNum)

Parameters  -> btLibRefNum  Reference number for the Bluetooth library.

Result  Returns one of the following values:

btLibErrNoError  Success.

btLibErrBusy  The piconet is in the process of being destroyed.

btLibErrNoPiconet  No piconet exists.

btLibErrNotOpen  The referenced Bluetooth library is not open.

btLibErrStackNotOpen  The Bluetooth stack failed to initialize when the library was opened.

Comments  After locking inbound connections, outbound connections are still allowed. Locking inbound connections maximizes the bandwidth for members of the piconet to transmit data to each other.

See Also  BtLibPiconetUnlockInbound
New

BtLibPiconetUnlockInbound

Purpose
Allow remote devices to create ACL links into the piconet.

Declared In
BtLib.h

Prototype
Err BtLibPiconetUnlockInbound
(UInt16 btLibRefNum, Boolean discoverable)

Parameters
-> btLibRefNum  Reference number for the Bluetooth library.
-> discoverable
  If true, configures the radio to be discoverable.
  In other words, the radio responds to inquiries.
  If false, configures the radio to be only connectable.
  In other words, only devices that know the radio’s Bluetooth device address can connect to it.

Result
Returns one of the following values:

  btLibErrNoError
  Success.

  btLibErrBusy
  The piconet is in the process of being destroyed.

  btLibErrNoPiconet
  No piconet exists.

  btLibErrNotOpen
  The referenced Bluetooth library is not open.

  btLibErrStackNotOpen
  The Bluetooth stack failed to initialize when the library was opened.
Bluetooth Library: Management
Management Functions

Comments
Allowing inbound connections lowers the bandwidth available to transmit data between members of the piconet because the radio must periodically scan for incoming links.

See Also
BtLibPiconetLockInbound

New
BtLibRegisterManagementNotification

Purpose
Register a callback function to process events generated by management functions.

Declared In
BtLib.h

Prototype
Err BtLibRegisterManagementNotification
(UInt16 btLibRefNum,
BtLibManagementProcPtr callbackP, UInt32 refCon)

Parameters
-> btLibRefNum
Reference number for the Bluetooth library.

-> callbackP
Pointer to a callback procedure to register. This pointer must not be NULL. See BtLibManagementCallback for more information.

-> refCon
Application-defined data to pass to the event handler.

Result
Returns one of the following values:

btLibErrNoError
Success.

btLibErrAlreadyRegistered
The callback has already been registered with the management entity.

btLibErrNotOpen
The referenced Bluetooth library is not open.

btLibErrParamError
One or more parameters is invalid.
Bluetooh Library: Management

Management Functions

btLibErrStackNotOpen
The Bluetooth stack failed to initialize when the library was opened.

btLibErrTooMany
There is no space available to store the callback. The management entity allows a maximum of 16 callbacks to be registered at a time.

Comments
The management functions are asynchronous. That is, they return immediately and generate events when the task actually completes at a later time. To handle these events, you need to define a callback function with the same prototype as BtLibManagementCallback. Then you need to register your callback function using BtLibRegisterManagementNotification. For examples of the callback events your callback function needs to handle, see the Management Callback Events section.

Applications should unregister their management callbacks before closing the Bluetooth library to prevent the callback table from overflowing. The callback table holds a maximum of 16 entries.

See Also
BtLibUnregisterManagementNotification

New
BtLibSetGeneralPreference

Purpose
Set one of the general management preferences.

Declared In
BtLib.h

Prototype
Err BtLibSetGeneralPreference
(UInt16 btLibRefNum, BtLibGeneralPrefEnum pref,
void *prefValue, UInt16 prefValueSize)

Parameters
- -> btLibRefNum Reference number for the Bluetooth library.
- -> pref General preference to set. See BtLibGeneralPreferenceEnum.
Bluetooth Library: Management

Management Functions

-> prefValue  Pointer to the value of the preference. This parameter must not be NULL. See BtLibGeneralPreferenceEnum.

-> prefValueSize  Size, in bytes, of prefValue.

Result  Returns one of the following values:

btLibErrNoError  Success.

btLibErrPending  The results will be returned through a notification.

btLibErrNotOpen  The referenced Bluetooth library is not open.

btLibErrParamError  One or more parameters is invalid. Be sure that prefValueSize matches the size of the preference value.

btLibErrStackNotOpen  The Bluetooth stack failed to initialize when the library was opened.

Comments  See the BtLibGetGeneralPreference function description for a list of the preferences.

This function may generate events depending on the preference you change. The btLibManagementEventAccessibilityChange event indicates that the accessibility of the local device has changed.

See Also  BtLibGetGeneralPreference
**New**

**BtLibStartInquiry**

**Purpose**
Start a Bluetooth inquiry.

**Declared In**
BtLib.h

**Prototype**
Err BtLibStartInquiry (UInt16 btLibRefNum, UInt8 timeout, UInt8 maxResp)

**Parameters**
- `btLibRefNum` Reference number for the Bluetooth library.
- `timeout` Time, in seconds, this inquiry is allowed to take. If the inquiry does not complete within this time, it is canceled. The actual time is rounded to the nearest multiple of 1.28 seconds. If you specify a timeout period larger than 60 seconds, this function acts as if you specified a timeout period of 60 seconds. If this parameter is 0, the timeout period defaults to 10.24 seconds as specified in the Generic Access Profile.
- `maxResp` Maximum number of responses the inquiry accepts. Responses are not guaranteed to be unique.

**Result**
Returns one of the following values:

- `btLibErrPending` The results will be returned through callback events.
- `btLibErrBluetoothOff` The Bluetooth radio is off. The user can turn the radio on and off with a setting in the preferences panel.
- `btLibErrInProgress` Another inquiry is already in progress.
Bluetooth Library: Management

Management Functions

btLibErrNotOpen
The referenced Bluetooth library is not open.

btLibErrStackNotOpen
The Bluetooth stack failed to initialize when the library was opened.

Comments
The function performs a low-level Bluetooth inquiry, as opposed to a full device discovery. Specifically, inquiries started with this function only return the Bluetooth address and the class of the discovered device. This function does not have a user interface.

Every time a device is discovered, a btLibManagementEventInquiryResult callback event is generated. When the inquiry is complete, a btLibManagementEventInquiryComplete callback event is generated. If the application calls BtLibCancelInquiry, a btLibManagementEventInquiryCanceled callback event is generated.

See Also  BtLibCancelInquiry

New  BtLibUnregisterManagementNotification

Purpose
Unregister a previously registered management callback.

Declared In  BtLib.h

Prototype
Err BtLibUnregisterManagementNotification
(UInt16 btLibRefNum,  
BtLibManagementProcPtr callbackP)

Parameters
-> btLibRefNum  Reference number for the Bluetooth library.
-> callbackP    Pointer to the callback procedure to unregister. This pointer must not be NULL.

Result
Returns one of the following values:
btLibErrNoError
   Success.

btLibErrError
   The callback referenced by callbackP has not been registered.

btLibErrNotOpen
   The referenced Bluetooth library is not open.

btLibErrParamError
   One or more parameters are invalid.

btLibErrStackNotOpen
   The Bluetooth stack failed to initialize when the library was opened.

Comments
   Applications should unregister their management callbacks before closing the library to prevent the callback table from overflowing. The callback table holds a maximum of 16 entries.

See Also
   BtLibRegisterManagementNotification

Application-Defined Functions

   This section describes the callback functions that handle management events. These functions are supplied by the developer and can be named anything.
New **BtLibManagementCallback**

**Purpose** Signal the result of a Bluetooth management event. When the event takes place, this callback function is called.

**Declared In** BtLibTypes.h

**Prototype**

```c
void (*BtLibManagementProcPtr)(BtLibManagementEventType *mEvent, UInt32 refCon)
```

**Parameters**

- `mEvent` *BtLibManagementEventType* structure containing the event parameters.
- `refCon` General purpose integer which you can use to hold application-specific information. When you register the callback with the `BtLibRegisterManagementNotification` function, you can specify a value to pass to this parameter.

**Result** Returns nothing.

**Comments**

The event and status of the event are in the `mEvent` structure. See [Management Callback Events](#) for more information.

You must register this function using the `BtLibRegisterManagementNotification` function before it starts receiving events.
Bluetooth Library:
Sockets and Service Discovery

The Bluetooth library uses sockets to represent L2CAP, RFCOMM, and SDP connections. This chapter presents reference material for the socket and SDP support provided by the Bluetooth library API:

**Socket-Related Data Structures**

This section lists some of the more important types used by the Bluetooth library.

**Socket Callback Events**

The Bluetooth library socket API supports L2CAP, RFCOMM, and SDP, the upper protocols of the Bluetooth specification. As with the management functions, the socket functions are mostly asynchronous. To signal to the application that socket operations have completed, the Bluetooth library generates *socket callback events* by calling a callback function.

**Socket Disconnection Error Codes**

In addition to the standard error codes that can accompany socket events, the status codes accompanying the disconnection and connection events can have the additional values enumerated in this section.

**Socket Functions**

The functions in this section perform general socket tasks and tasks related to L2CAP and RFCOMM sockets. The functions specific to SDP sockets are in the next section.
Bluetooth Library: Sockets and Service Discovery

Socket-Related Data Structures

This section describes functions and macros specific to the Bluetooth Service Discovery Protocol (SDP).

Application-Defined Functions

This section describes the callback functions that handle socket events.

The header file BtLib.h declares the Bluetooth library functions and macros. The header file BtLibTypes.h declares the data structures that you use with those functions and macros. For more information about using the Bluetooth library, see the Palm OS Programmer’s Companion Supplement: Bluetooth.

Socket-Related Data Structures

This section lists some of the more important types used by the Bluetooth library.

New

BtLibL2CapPsmType

The BtLibL2CapPsmType type represents a Protocol and Server Multiplexer (PSM) value. See the “Logical Link and Adaptation Protocol Specification” chapter of the Specification of the Bluetooth System for more information about PSM values. The Bluetooth library only supports two-byte PSM values.

typedef UInt16 BtLibL2CapPsmType;

New

BtLibLanguageBaseTripletType

The BtLibLanguageBaseTripletType structure represents a language base attribute identifier list attribute. See the “Service Discovery Protocol” chapter of the Specification of the Bluetooth System for more information.

typedef struct BtLibLanguageBaseTripletType {
  UInt16 naturalLanguageIdentifier;
} BtLibLanguageBaseTripletType;
field Descriptions

naturalLanguageIdentifier
A UInt16 representing a natural language. See Language Constants for a set of constants that can be used in this field.

characterEncoding
A UInt16 representing a character set encoding. See Character Encoding Constants for a set of constants that can be used in this field.

baseAttributeID
Base attribute identifiers for attributes represented in this language. See Attribute Identifier Offsets for offsets that are added to this value to get the attribute identifiers for specific attributes represented in this language.

Language Constants
These constants are used in the naturalLanguageIdentifier field of the BtLibLanguageBaseTripletType and are defined in the ISO 639:1988 specification.

btLibLangAfar
btLibLangAbkihazian
btLibLangAfrikaans
btLibLangAmharic
btLibLangArabic
btLibLangAssamese
btLibLangAymara
btLibLangAzerbaijani
btLibLangBashkir
btLibLangByelorussian
btLibLangBulgarian
btLibLangBihari
btLibLangBislama
btLibLangBengali
btLibLangTibetan
btLibLangBreton
btLibLangCatalan
btLibLangCorsican
btLibLangCzech
btLibLangWelsh
btLibLangDanish
btLibLangGerman
btLibLangBhutani
btLibLangGreek
btLibLangEnglish
btLibLangEsperanto
btLibLangSpanish
btLibLangEstonian
btLibLangBasque
btLibLangPersian
btLibLangFinnish
btLibLangFiji
btLibLangFaroese
btLibLangFrench
btLibLangFrisian
btLibLangIrish
btLibLangScotsGaelic
btLibLangGalician
btLibLangGuarani
btLibLangGujarati
btLibLangHausa
btLibLangHindi
btLibLangCroatian
btLibLangHungarian
btLibLangArmenian
btLibLangInterlingua
btLibLangInterlingue
btLibLangInupiaq
btLibLangIndonesian
btLibLangIcelandic
btLibLangItalian
btLibLangHebrew
btLibLangJapanese
btLibLangYiddish
btLibLangJavanese
btLibLangGeorgian
btLibLangKazakh
btLibLangGreenlandic
btLibLangCambodian
btLibLangKannada
btLibLangKorean
btLibLangKashmiri
btLibLangKurdish
btLibLangKirghiz
btLibLangLatin
btLibLangLingala
btLibLangLaotian
btLibLangLithuanian
btLibLangLatvian
btLibLangMalagasy
btLibLangMaori
btLibLangMacedonian
btLibLangMalayalam
btLibLangMongolian
btLibLangMoldavian
btLibLangMarathi
btLibLangMalay
btLibLangMaltese
btLibLangBurmese
btLibLangNauru
btLibLangNepali
btLibLangDutch
btLibLangNorwegian
btLibLangOccitan
btLibLangOromo
btLibLangOriya
btLibLangPunjabi
btLibLangPolish
btLibLangPashto
btLibLangPortuguese
btLibLangQuechua
btLibLangRhaeto_Romance
btLibLangKirundi
Bluetooth Library: Sockets and Service Discovery
Socket-Related Data Structures

btLibLangRomanian
btLibLangRussian
btLibLangKinyarwanda
btLibLangSanskrit
btLibLangSindhi
btLibLangSangho
btLibLangSerbo_Croatian
btLibLangSinghalese
btLibLangSlovak
btLibLangSlovenian
btLibLangSamoan
btLibLangShona
btLibLangSomali
btLibLangAlbanian
btLibLangSerbian
btLibLangSiswati
btLibLangSesotho
btLibLangSundanese
btLibLangSwedish
btLibLangSwahili
btLibLangTamil
btLibLangTelugu
btLibLangTajik
btLibLangThai
btLibLangTigrinya
btLibLangTurkmen
btLibLangTagalog
btLibLangSetswana
btLibLangTonga
btLibLangTurkish
btLibLangTsonga
btLibLangTatar
btLibLangTwi
btLibLangUkrainian
btLibLangUrdu
btLibLangUzbek
btLibLangVietnamese
btLibLangVolapuk
btLibLangWolof
btLibLangXhosa
Character Encoding Constants

These constants are used to specify the character encoding used for SDP attributes. More information about these character sets can be found at http://www.iana.org/assignments/character-sets.

- btLibCharSet_US_ASCII
- btLibCharSet_Adobe_Standard_Encoding
- btLibCharSet_Adobe_Symbol_Encoding
- btLibCharSet_ANSI_X3_110_1983
- btLibCharSet_ASMO_449
- btLibCharSet_Big5
- btLibCharSet_Big5_HKSCS
- btLibCharSet_BS_4730
- btLibCharSet_BS_viewdata
- btLibCharSet_CSA_Z243_4_1985_1
- btLibCharSet_CSA_Z243_4_1985_2
- btLibCharSet_CSA_Z243_4_1985_gr
- btLibCharSet_CSN_369103
- btLibCharSet_DEC_MCS
- btLibCharSet_DIN_66003
- btLibCharSet_dk_us
- btLibCharSet_DS_2089
- btLibCharSet_EBCDIC_AT_DE
- btLibCharSet_EBCDIC_AT_DE_A
- btLibCharSet_EBCDIC_CA_FR
- btLibCharSet_EBCDIC_DK_NO
- btLibCharSet_EBCDIC_DK_NO_A
- btLibCharSet_EBCDIC_ES
- btLibCharSet_EBCDIC_ES_A
- btLibCharSet_EBCDIC_ES_S
- btLibCharSet_EBCDIC_FI_SE
- btLibCharSet_EBCDIC_FI_SE_A
- btLibCharSet_EBCDIC_FR
- btLibCharSet_EBCDIC_IT
- btLibCharSet_EBCDIC_PT
btLibCharSet_EBCDIC_UK
btLibCharSet_EBCDIC_US
btLibCharSet_ECMA_cyrillic
btLibCharSet_ES 23
btLibCharSet_ES2
btLibCharSet_EUC_JP
btLibCharSet_EUC_KR
btLibCharSet_Extended_UNIX_Code_Fixed_Width_for_Japanese
btLibCharSet_GB2312
btLibCharSet_GB_1988_80
btLibCharSet_GB_2312_80
btLibCharSet_GOST_19768_74
btLibCharSet_greek7
btLibCharSet_greek7_old
btLibCharSet_greek_ccitt
btLibCharSet_HP_DeskTop
btLibCharSet_HP_Legal
btLibCharSet_HP_Math8
btLibCharSet_HP_Pi_font
btLibCharSet_hp_roman8
btLibCharSet_HZ_GB_2312
btLibCharSet_IBM037
btLibCharSet_IBM038
btLibCharSet_IBM273
btLibCharSet_IBM274
btLibCharSet_IBM275
btLibCharSet_IBM277
btLibCharSet_IBM278
btLibCharSet_IBM280
btLibCharSet_IBM281
btLibCharSet_IBM284
btLibCharSet_IBM285
btLibCharSet_IBM290
btLibCharSet_IBM297
btLibCharSet_IBM420
btLibCharSet_IBM423
btLibCharSet_IBM424
btLibCharSet_IBM437
btLibCharSet_IBM500
btLibCharSet_IBM775
btLibCharSet_IBM850
btLibCharSet_IBM851
btLibCharSet_IBM852
btLibCharSet_IBM855
btLibCharSet_IBM857
btLibCharSet_IBM860
btLibCharSet_IBM861
btLibCharSet_IBM862
btLibCharSet_IBM863
btLibCharSet_IBM864
btLibCharSet_IBM865
btLibCharSet_IBM866
btLibCharSet_IBM868
btLibCharSet_IBM869
btLibCharSet_IBM870
btLibCharSet_IBM871
btLibCharSet_IBM880
btLibCharSet_IBM891
btLibCharSet_IBM903
btLibCharSet_IBM904
btLibCharSet_IBM905
btLibCharSet_IBM918
btLibCharSet_IBM1026
btLibCharSet_IBM00858
btLibCharSet_IBM00924
btLibCharSet_IBM01140
btLibCharSet_IBM01141
btLibCharSet_IBM01142
btLibCharSet_IBM01143
btLibCharSet_IBM01144
btLibCharSet_IBM01145
btLibCharSet_IBM01146
btLibCharSet_IBM01147
btLibCharSet_IBM01148
btLibCharSet_IBM01149
btLibCharSet_IBM_Symbols
btLibCharSet_IBM_Thai
btLibCharSet_IEC_P27_1
btLibCharSet_INIS
btLibCharSet_INIS_8
btLibCharSet_INIS_cyrillic
btLibCharSet_INVARIANT
btLibCharSet_ISO_646_basic_1983
btLibCharSet_ISO_646_irv_1983
btLibCharSet_ISO_2022_CN
btLibCharSet_ISO_2022_CN_EXT
btLibCharSet_ISO_2022_JP
btLibCharSet_ISO_2022_JP_2
btLibCharSet_ISO_2022_KR
btLibCharSet_ISO_2033_1983
btLibCharSet_ISO_5427
btLibCharSet_ISO_5427_1981
btLibCharSet_ISO_5428_1980
btLibCharSet_ISO_6937_2_25
btLibCharSet_ISO_6937_2_add
btLibCharSet_ISO_8859_1
btLibCharSet_ISO_8859_10
btLibCharSet_iso_8859_13
btLibCharSet_iso_8859_14
btLibCharSet_ISO_8859_15
btLibCharSet_ISO_8859_1_Windows_3_0_Latin_1
btLibCharSet_ISO_8859_1_Windows_3_1_Latin_1
btLibCharSet_ISO_8859_2
btLibCharSet_ISO_8859_2_Windows_Latin_2
btLibCharSet_ISO_8859_3
btLibCharSet_ISO_8859_4
btLibCharSet_ISO_8859_5
btLibCharSet_ISO_8859_6
btLibCharSet_ISO_8859_6_E
btLibCharSet_ISO_8859_6_I
btLibCharSet_ISO_8859_7
btLibCharSet_ISO_8859_8
btLibCharSet_ISO_8859_8_E
btLibCharSet_ISO_8859_8_I
btLibCharSet_ISO_8859_9
btLibCharSet_ISO_8859_9_Windows_Latin_5
btLibCharSet_ISO_8859_supp
btLibCharSet_ISO_10367_box
btLibCharSet_ISO_10646_UCS_2
Bluetooth Library: Sockets and Service Discovery

Socket-Related Data Structures

btLibCharSet_ISO_10646_UCS_4
btLibCharSet_ISO_10646_UCS_Basic
btLibCharSet_ISO_10646_Unicode_Latin1
btLibCharSet_ISO_10646_UTF_1
btLibCharSet_iSO_ir_90
btLibCharSet_ISO_Unic ode_IBM_1261
btLibCharSet_ISO_Unic ode_IBM_1264
btLibCharSet_ISO_Unic ode_IBM_1265
btLibCharSet_ISO_Unic ode_IBM_1268
btLibCharSet_ISO_Unic ode_IBM_1276
btLibCharSet_IT
btLibCharSet_JIS_C6220_1969_jp
btLibCharSet_JIS_C6220_1969_ro
btLibCharSet_JIS_C6226_1978
btLibCharSet_JIS_C6226_1983
btLibCharSet_JIS_C6229_1984_a
btLibCharSet_JIS_C6229_1984_b
btLibCharSet_JIS_C6229_1984_b_add
btLibCharSet_JIS_C6229_1984_hand
btLibCharSet_JIS_C6229_1984_hand_add
btLibCharSet_JIS_C6229_1984_kana
btLibCharSet_JIS_Encoding
btLibCharSet_JIS_X0201
btLibCharSet_JIS_X0212_1990
btLibCharSet_JUS_I_B1_002
btLibCharSet_JUS_I_B1_003_mac
btLibCharSet_JUS_I_B1_003_serb
btLibCharSet_KOI8_R
btLibCharSet_KOI8_U
btLibCharSet_KSC5636
btLibCharSet_KS_C_5601_1987
btLibCharSet_latin_greek
btLibCharSet_Latin_greek_1
btLibCharSet_latin_lap
btLibCharSet_macintosh
btLibCharSet_Microsoft_Publishing
btLibCharSet_MNEM
btLibCharSet_MNEMONIC
btLibCharSet_MSZ_7795_3
btLibCharSet_NATS_DANO
btLibCharSet_NATS_DANO_ADD
btLibCharSet_NATS_SEFI
btLibCharSet_NATS_SEFI_ADD
btLibCharSet_NC_NC00_10_81
btLibCharSet_NF_Z_62_010
btLibCharSet_NF_Z_62_010__1973_
bLibCharSet_NS_4551_1
btLibCharSet_NS_4551_2
btLibCharSet_PC8_Danish_Norwegian
btLibCharSet_PC8_Turkish
btLibCharSet_PT
btLibCharSet_PT2
btLibCharSet_SCSU
btLibCharSet_SEN_850200_B
btLibCharSet_SEN_850200_C
btLibCharSet_Shift_JIS
btLibCharSet_TIS_620
btLibCharSet_T_101_G2
btLibCharSet_T_61_7bit
btLibCharSet_T_61_8bit
btLibCharSet_UNICODE_1_1
btLibCharSet_UNICODE_1_1_UTF_7
btLibCharSet_UNKNOWN_8BIT
btLibCharSet_us_dk
btLibCharSet_UTF_16
btLibCharSet_UTF_16BE
btLibCharSet_UTF_16LE
btLibCharSet_UTF_7
btLibCharSet_UTF_8
btLibCharSet_Ventura_International
btLibCharSet_Ventura_Math
btLibCharSet_Ventura_US
btLibCharSet_videotex_suppl
btLibCharSet_VIQR
btLibCharSet_VISCII
btLibCharSet_Windows_31J
btLibCharSet_windows_1250
btLibCharSet_windows_1251
btLibCharSet_windows_1252
btLibCharSet_windows_1253
Attribute Identifier Offsets

btLibServiceNameOffset
The offset from the baseAttributeID to get the attribute identifier of the service name.

btLibServiceDescriptionOffset
The offset from the baseAttributeID to get the attribute identifier of the service description.

btLibProviderNameOffset
The offset from the baseAttributeID to get the attribute identifier of the provider name.

New

BtLibProfileDescriptorListEntryType

The BtLibProfileDescriptorListEntryType structure represents an entry in a profile descriptor list attribute. See the “Service Discovery Protocol” chapter of the Specification of the Bluetooth System for more information about profile descriptor list attributes.

typedef struct
BtLibProfileDescriptorListEntryType {

BtLibSdpUuidType profUUID;
UInt16 version;
} BtLibProfileDescriptorListEntryType;

Field Descriptions

profUUID The UUID of the profile.
version The version of the profile.
**New** BtLibProtocolDescriptorListEntryType

The BtLibProtocolDescriptorListEntryType structure represents an entry in a protocol descriptor list attribute. See the “Service Discovery Protocol” chapter of the Specification of the Bluetooth System for more information.

```c
typedef struct

  BtLibProtocolDescriptorListEntryType {

  BtLibSdpUuidType protoUUID;
  union {
    BtLibL2CapPsmType psm;
    BtLibRfCommServerIdType channel;
  } param;
} BtLibProtocolDescriptorListEntryType;
```

***Field Descriptions***

- **protoUUID**
  The UUID of the protocol.

- **param**
  A union containing two members: `psm` and `channel`. `psm` is applicable for a L2CAP protocol descriptor and specifies the Protocol and Service Multiplexor. `channel` is applicable to a RFCOMM protocol descriptor and specifies the server channel.

---

**New** BtLibRfCommServerIdType

The BtLibRfCommServerIdType type represents a RFCOMM server channel. See the “RFCOMM with TS 07.10” chapter of the Specification of the Bluetooth System for more information about server channels.
typedef UInt8 BtLibRfCommServerIdType;

---

**New**

BtLibSdpAttributeDataType

The BtLibSdpAttributeDataType union is used to encapsulate an SDP attribute or a list entry in an SDP attribute. The BtLibSdpServiceRecordGetAttribute function gets an attribute or a list entry and return its contents in a BtLibSdpAttributeDataType. The BtLibSdpServiceRecordSetAttribute function sets an attribute or list entry according to the contents of a BtLibSdpAttributeDataType. This type supports the universal attributes defined in the *Specification of the Bluetooth System*.

```c
typedef union BtLibSdpAttributeDataType {
    BtLibSdpUuidType serviceClassUuid;
    UInt32 serviceRecordState;
    BtLibSdpUuidType serviceIdUuid;
    BtLibProtocolDescriptorListEntryType protocolDescriptorListEntry;
    BtLibSdpUuidType browseGroupUuid;
    BtLibLanguageBaseTripletType languageBaseTripletListEntry;
    UInt32 timeToLive;
    UInt8 availability;
    BtLibProfileDescriptorListEntryType profileDescriptorListEntry;
    BtLibUrlType documentationUrl;
    BtLibUrlType clientExecutableUrl;
    BtLibUrlType iconUrl;
    BtLibStringType serviceName;
    BtLibStringType providerName;
} BtLibSdpAttributeDataType;
```

Note that if you’re retrieving a string or a URL using the BtLibSdpServiceRecordGetAttribute function, you first need to allocate a buffer in addition to this union. This buffer must be large enough to contain the anticipated size of the string or URL. You must also initialize the string pointer and string length fields of the appropriate BtLibAttributeDataType union member. For
example, if you’re retrieving an icon URL, you need to set iconURL.url to point to the buffer. You also need to set iconURL.urllen to the length of the buffer.

See Also

BtLibSdpUuidType,
BtLibProtocolDescriptorListEntryType,
BtLibProfileDescriptorListEntryType,
BtLibLanguageBaseTripletType, BtLibUrlType,
BtLibStringType

New

BtLibSdpAttributeIdType

The BtLibSdpAttributeIdType type represents a SDP attribute identifier.

typedef UInt16 BtLibSdpAttributeIdType;

The following constants are defined by the Bluetooth library. They represent the universal attributes in the Specification of the Bluetooth System.

Universal Attribute IDs

btLibServiceClassIdList
btLibServiceRecordState
btLibServiceId
btLibProtocolDescriptorList
tbLibBrowseGroupList
btLibLanguageBaseAttributeIdList
tbLibTimeToLive
btLibAvailability
btLibProfileDescriptorList
btLibDocumentationUrl
btLibClientExecutableUrl
btLibIconUrl
New **BtLibSdpRecordHandle**

The `BtLibSdpRecordHandle` type, also called an SDP memory handle, provides a memory handle to an SDP memory record.

```c
typedef MemHandle BtLibSdpRecordHandle;
```

A SDP memory record can have two roles: it can contain a local SDP service record or it can refer to an SDP service record on a remote device. In the latter role, the SDP memory record is said to be mapped to a service record on the remote device. The `BtLibSdpServiceRecordMapRemote` function performs this mapping.

New **BtLibSdpRemoteServiceRecordHandle**

The `BtLibSdpRemoteServiceRecordHandle` type represents a SDP service record handle on a remote device as defined in the “Service Discovery Protocol” chapter of the *Specification of the Bluetooth System*. The documentation refers to this type as a remote service record handle.

```c
typedef UInt32 BtLibSdpRemoteServiceRecordHandle;
```

Note that this type is different from the `BtLibSdpRecordHandle` type, which refers to a memory chunk containing an SDP service record.

New **BtLibSdpUuidSizeEnum**

The `BtLibSdpUuidSizeEnum` enum specifies the sizes that a UUID can have. See `BtLibSdpUuidType` for more information.

```c
typedef enum {
    btLibUuidSize16 = 2,
    btLibUuidSize32 = 4,
    btLibUuidSize128 = 16
} BtLibSdpUuidSizeEnum;
```
Value Descriptions

btLibUuidSize16
16-bit UUID

btLibUuidSize32
32-bit UUID

btLibUuidSize128
Full size 128-bit UUID

New BtLibSdpUuidType

The BtLibSdpUuidType structure represents a Universally Unique Identifier (UUID). A UUID is a 128-bit value that is generated in a manner that guarantees that it is different from every other UUID.

The “Service Discovery Protocol” chapter of the Specification of the Bluetooth System reserves a set of UUIDs for common Bluetooth services and protocols. You can specify these with 32 bits—the remaining 96 bits have a fixed value. A subset of these can be specified with 16 bits zero-extended to 32 bits. Therefore you can specify a UUID using 16, 32, or 128 bits.

You generally don’t set this type directly. Instead, you use the BtLibSdpUuidInitialize macro.

typedef struct BtLibSdpUuidType {
    BtLibSdpUuidSizeEnum size;
    UInt8 UUID[16];
} BtLibSdpUuidType;

Field Descriptions

size The number of bits used to specify the UUID. See BtLibSdpUuidSizeEnum.

UUID The value of the UUID. If you’re setting the value of this field, use the BtLibSdpUuidInitialize macro.
Predefined UUIDs

btLibSdpUUID_SC_SERVICE_DISCOVERY_SERVER
btLibSdpUUID_SC_BROWSE_GROUP_DESC
btLibSdpUUID_SC_PUBLIC_BROWSE_GROUP
btLibSdpUUID_SC_SERIAL_PORT
btLibSdpUUID_SC_LAN_ACCESS_PPP
btLibSdpUUID_SC_DIALUP_NETWORKING
btLibSdpUUID_SC_IRMC_SYNC
btLibSdpUUID_SC_OBEX_OBJECT_PUSH
btLibSdpUUID_SC_OBEX_FILE_TRANSFER
btLibSdpUUID_SC_IRMC_SYNC_COMMAND
btLibSdpUUID_SC_HEADSET
btLibSdpUUID_SC_CORDLESS_TELEPHONY
btLibSdpUUID_SC_INTERCOM
btLibSdpUUID_SC_FAX
btLibSdpUUID_SC_HEADSET_AUDIO_GATEWAY
btLibSdpUUID_SC_WAP
btLibSdpUUID_SC_WAP_CLIENT
btLibSdpUUID_SC_PNP_INFO
btLibSdpUUID_SC_GENERIC_NETWORKING
btLibSdpUUID_SC_GENERIC_FILE_TRANSFER
btLibSdpUUID_SC_GENERIC_AUDIO
btLibSdpUUID_SC_GENERIC_TELEPHONY

btLibSdpUUID_PROT_SDP
btLibSdpUUID_PROT_RFCOMM
btLibSdpUUID_PROT_TCS_BIN
btLibSdpUUID_PROT_L2CAP
btLibSdpUUID_PROT_IP
btLibSdpUUID_PROT_UDP
btLibSdpUUID_PROT_TCP
btLibSdpUUID_PROT_TCS_AT
btLibSdpUUID_PROT_OBEX
btLibSdpUUID_PROT_FTP
btLibSdpUUID_PROT_HTTP
btLibSdpUUID_PROT_WSP
**New**  
**BtLibSocketEventType**

The `BtLibSocketEventType` structure contains detailed information regarding a socket callback event. All socket events have some common data. Most socket events have additional data specific to those events. The specific data is stored in a union that is part of the `BtLibSocketEvent` data structure.

```c
typedef struct _BtLibSocketEventType {
    BtLibSocketEventEnum event;
    BtLibSocketRef socket;
    Err status;
    union
    {
        ...
    } eventData;
} BtLibSocketEventType,
*BtLibSocketEventTypePtr;
```

**Field Descriptions**

- **event**  
  `BtLibSocketEventEnum` enum member that indicates which socket event has occurred. See [Socket Callback Events](#).

- **socket**  
  Socket associated with the event.

- **status**  
  Status of the event. The [Socket Callback Events](#) section gives more details about how to interpret this field for specific events.

- **eventData**  
  Data associated with the event. The member of this union that is valid depends on the event. See [Socket Callback Events](#) for more information.

**New**  
**BtLibSocketRef**

The `BtLibSocketRef` type identifies a socket.
typedef Int16 BtLibSocketRef;

**New**

<table>
<thead>
<tr>
<th><strong>BtLibStringType</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Field Descriptions</strong></td>
</tr>
<tr>
<td><strong>str</strong></td>
</tr>
<tr>
<td><strong>strLen</strong></td>
</tr>
</tbody>
</table>

**New**

<table>
<thead>
<tr>
<th><strong>BtLibUrlType</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Field Descriptions</strong></td>
</tr>
<tr>
<td><strong>url</strong></td>
</tr>
<tr>
<td><strong>urlLen</strong></td>
</tr>
</tbody>
</table>

**Socket Callback Events**

The Bluetooth library socket API supports L2CAP, RFCOMM, and SDP, the upper protocols of the Bluetooth specification. As with the
management functions, the socket functions are mostly asynchronous. To signal to the application that socket operations have completed, the Bluetooth library generates a socket callback events by calling a callback function.

You specify the callback function when you create a socket. When an event occurs, the callback function is called with two parameters: a pointer to a BtLibSocketEventType structure and a pointer to a user-defined structure.

The BtLibSocketEventType structure contains an event field, which indicates the reason the callback is called, a status field, which indicates status information associated with the event, a socket field, which indicates the socket associated with the event, and a union of several structures. The member of the union that is valid depends on the event. The meaning of the events is described in the following sections.

**btLibSocketEventConnectedInbound**

A remote connection has been accepted because the application has called BtLibSocketRespondToConnection.

For this event, the eventData field contains the following field:

```
BtLibSocketRef newSocket;
```

This BtLibSocketRef contains the reference to the new socket.

If the remote device requests a L2CAP connection, this event is sent to the L2CAP listener socket with a PSM that matches the PSM of the requested connection. The Bluetooth library creates a new socket that exchanges data with the remote device.

If the remote device requests an RFCOMM connection, this event is sent to the RFCOMM listener socket with a server channel that matches the server channel of the requested connection. The Bluetooth library converts the listener socket into a socket that exchanges data with the remote device.

**btLibSocketEventConnectedOutbound**

An outbound connection initiated by a call to BtLibSocketConnect has completed. The status field is
btLibErrNoError if the connection has completed successfully. Otherwise, the status field indicates why the connection failed.

**btLibSocketEventConnectRequest**

A remote device has requested a connection.

You must respond to this event with a call to `BtLibSocketRespondToConnection`.

For this event, the `eventData` field contains the following field:

```c
BtLibDeviceAddressType requestingDevice;
```

This `BtLibDeviceAddressType` contains the address of the remote device requesting the connection.

If the remote device requests a L2CAP connection, this event is sent to the L2CAP listener socket with a PSM that matches the PSM of the request.

If the remote device requests an RFCOMM connection, this event is sent to the RFCOMM listener socket with a server channel that matches the server channel of the request.

To convert a socket into a listener socket use the `BtLibSocketListen` function.

**btLibSocketEventData**

Data has been received on a socket.

For this event, the `eventData` field contains the following structure:

```c
struct {
    Uint16 dataLen;
    Uint8 *data;
} data;
```

**Value Descriptions**

- **dataLen**: The size, in bytes, of the received data.
- **data**: A pointer to the received data.
Bluetooth Library: Sockets and Service Discovery

Socket Callback Events

**btLibSocketEventDisconnected**
The connection has been lost or one of the devices has disconnected. The socket is now invalid. The status field indicates the reason for the disconnection.

**btLibSocketEventSdpServiceRecordHandle**
A request for remote service records matching a list of service classes has completed. The application initiated this request by calling the *BtLibSdpServiceRecordsGetByServiceClass* function.

If the status field is btLibErrNoError, the SDP operation completed successfully, and the eventData field contains valid information. Otherwise the SDP operation failed, and the status field indicates the reason for the failure.

For this event, the eventData field contains the following structure:

```
struct {
    UInt16 numSrvRec;
    BtLibSdpRemoteServiceRecordHandle *serviceRecordList;
} sdpServiceRecordHandles;
```

**Value Descriptions**

- **numSrvRec**
  Number of remote service record handles in the returned array.

- **serviceRecordList**
  An array of *BtLibSdpRemoteServiceRecordHandles* for the service records matching the service class list.

**btLibSocketEventSdpGetAttribute**
An attribute request has completed. The application initiated this request by calling the *BtLibSdpServiceRecordGetAttribute* function.
If the `status` field is `btLibErrNoError`, the operation completed successfully, and the `eventData` field contains valid data. Otherwise the operation failed, and the `status` field indicates the reason for the failure.

For this event, the `eventData` field contains the following structure:

```c
struct {
    BtLibSdpAttributeIdType attributeID;
    BtLibSdpRecordHandle recordH;
    union {
        ...
    } info;
} sdpAttribute;
```

**Value Descriptions**

- `attributeID`  The attribute identifier of the attribute.
- `recordH`  A handle identifying the SDP memory record from which the attribute is retrieved.
- `info`  A union containing information specific to the event. See *The info Field*.

**The info Field**

For this event, the `info` field contains the following structure:

```c
struct {
    BtLibSdpAttributeDataType *attributeValues;
    UInt16 listNumber;
    UInt16 listEntry;
} data;
```

**Value Descriptions**

- `attributeValues`  A `BtLibSdpAttributeDataType` containing the value of the attribute or list entry.
- `listNumber`  The index of the list in which this list entry appears or 0 if the attribute is not a protocol descriptor list. The index of the first list is 0.
listEntry

The index of the list entry within the list or 0 if the attribute is not a list. The index of the first entry is 0.

**btLibSocketEventSdpGetStringLen**

A string or URL length request has completed. The application initiated this request by calling `BtLibSdpServiceRecordGetStringOrUrlLength`.

If the `status` field is `btLibErrNoError`, the operation completed successfully, and length can be found in the `eventData` field. Otherwise the operation failed, and the `status` field indicates the reason for the failure.

For this event, the `eventData` field contains the following structure:

```c
struct {
    BtLibSdpAttributeIdType attributeID;
    BtLibSdpRecordHandle recordH;
    union {
        ...
    } info;
} sdpAttribute;
```

**Value Descriptions**

- **attributeID** The attribute identifier of the attribute.
- **recordH** A handle identifying the SDP memory record from which the string length is retrieved.
- **info** A union containing information specific to the event. See **The info Field**.

**The info Field**

For this event, the `info` field contains the following field:

```c
UInt16 strLength;
```

This field contains the length of the string or URL represented by the attribute. Bluetooth strings and URLs are not null-terminated.
btLibSocketEventSdpGetNumListEntries

A number of list entries request has completed. The application initiated this request by calling `BtLibSdpServiceRecordGetNumListEntries`.

If the status field is `btLibErrNoError`, the operation completed successfully, and the number of list entries can be found in the `eventData` field. Otherwise the operation failed, and the status field indicates the reason for the failure.

For this event, the `eventData` field contains the following structure:

```c
struct {
    BtLibSdpAttributeIdType attributeID;
    BtLibSdpRecordHandle recordH;
    union {
        ...
    } info;
} sdpAttribute;
```

**Value Descriptions**

- **attributeID**: The attribute identifier of the attribute.
- **recordH**: A handle identifying the SDP memory record from which the number of list entries is retrieved.
- **info**: A union containing information specific to the event. See **The info Field**.

**The info Field**

For this event, the `info` field contains the following field:

```c
UInt16 numItems;
```

This field contains the number of entries in the list attribute.

btLibSocketEventSdpGetNumLists

A number of lists request has completed. The application initiated this request by calling `BtLibSdpServiceRecordGetNumLists`.
If the status field is btLibErrNoError, the operation completed successfully, and the number of lists can be found in the eventData field. Otherwise the operation failed, and the status field indicates the reason for the failure.

For this event, the eventData field contains the following structure:

```c
struct {
    BtLibSdpAttributeIdType attributeID;
    BtLibSdpRecordHandle recordH;
    union {
        ...
    } info;
} sdpAttribute;
```

Value Descriptions

attributeID      The attribute identifier of the attribute.
recordH          A handle identifying the SDP memory record from which the number of lists is retrieved.
info             A union containing information specific to the event. See The info Field.

The info Field

For this event, the info field contains the following field:

```c
UInt16 numItems;
```

This field contains the number of lists in the protocol list descriptor attribute.

btLibSocketEventSdpGetRawAttribute

A get raw attribute request has completed. The application initiated the request by calling BtLibSdpServiceRecordGetRawAttribute.

If the status field is btLibErrNoError, the operation completed successfully, and the raw attribute can be found in the eventData field. Otherwise the operation failed, and the status field indicates the reason for the failure.
For this event, the `eventData` field contains the following structure:

```c
struct {
    BtLibSdpAttributeIdType attributeID;
    BtLibSdpRecordHandle recordH;
    union {
        ...
    } info;
} sdpAttribute;
```

**Value Descriptions**

- `attributeID`  
  The attribute identifier of the attribute.

- `recordH`  
  A handle identifying the SDP memory record from which the raw attribute is retrieved.

- `info`  
  A union containing information specific to the event. See [The info Field](#).

**The info Field**

For this event, the `info` field contains the following structure:

```c
struct {
    UInt16 valSize;
    UInt8 *value;
} rawData;
```

**Value Descriptions**

- `valSize`  
  Number of size, in bytes, of the raw attribute value.

- `value`  
  Byte array containing the raw attribute value.

### btLibSocketEventSdpGetRawAttributeSize

A get raw attribute size request has completed. The application initiated this request by calling `BtLibSdpServiceRecordGetSizeOfRawAttribute`.

If the `status` field is `btLibErrNoError`, the operation completed successfully, and the size of the attribute can be found in the
eventData field. Otherwise the operation failed, and the status field indicates the reason for the failure.

For this event, the eventData field contains the following structure:

```c
struct {
    BtLibSdpAttributeIdType attributeID;
    BtLibSdpRecordHandle recordH;
    union {
        ...
    } info;
} sdpAttribute;
```

**Value Descriptions**

- **attributeID** The attribute identifier of the attribute.
- **recordH** A handle identifying the SDP memory record from which the size of the raw attribute is retrieved.
- **info** A union containing information specific to the event. See The info Field.

**The info Field**

For this event, the info field contains the following field:

```c
UInt16 valSize;
```

This field contains the size, in bytes, of the raw attribute value.

**btLibSocketEventSdpGetServerChannelByUuid**

A get server channel request has completed. The application initiated this request by calling `BtLibSdpGetServerChannelByUuid`.

If the status field is `btLibErrNoError`, the operation completed successfully, and the server channel can be found in the eventData field. Otherwise, the operation failed, and the status field indicates the reason for the failure.
For this event, the `eventData` field contains the following structure:

```c
struct {
    BtLibSdpRemoteServiceRecordHandle remoteHandle;
    union {
        ...
    } param;
} sdpByUuid;
```

### Value Descriptions
- **remoteHandle** The handle for the remote SDP service record.
- **param** A union containing information that depends on the event. See **The param Field**.

### The param Field
For this event, the `param` field contains the following field:

```c
BtLibRfCommServerIdType channel;
```

This `BtLibRfCommServerIdType` contains the RFCOMM server channel represented by the SDP service record.

#### `btLibSocketEventSdpGetPsmByUuid`
A get PSM request has completed. The application initiated this request by calling `BtLibSdpGetPSMByUuid`.

If the status field is `btLibErrNoError`, the operation completed successfully, and the server channel can be found in the `eventData` field. Otherwise, the operation failed, and the `status` field indicates the reason for the failure.

For this event, the `eventData` field contains the following structure:

```c
struct {
    BtLibSdpRemoteServiceRecordHandle remoteHandle;
    union {
        ...
    } param;
```
Bluetooth Library: Sockets and Service Discovery

Socket Callback Events

```c
} sdByUuid;
```

Value Descriptions

remoteHandle The handle for the remote SDP service record.

param A union containing information that depends on the event. See The param Field.

The param Field

For this event, the param field contains the following field:

```c
BtLibL2CapPsmType psm;
```

This `BtLibL2CapPsmType` contains the PSM value of the L2CAP channel represented by the SDP service record.

btLibSocketEventSendComplete

A send request has completed. The application initiated this request by calling `BtLibSocketSend`.

For this event, the eventData field contains the following structure:

```c
struct {
    Uint16 dataLen;
    Uint8 *data;
} data;
```

Value Descriptions

dataLen The number of bytes of data that were actually sent.

data Not used. This variable does not contain any valid information.

Error Conditions

btLibErrNoError Success.

btLibErrNoAclLink No ACL Link.
Socket Disconnection Error Codes

In addition to the standard error codes that can accompany socket events, the status codes accompanying the `btLibSocketEventConnectedInbound`, `btLibSocketEventConnectedOutbound`, and `btLibSocketEventDisconnected` events can have the following additional values:

- `btLibL2DiscConfigOptions`  
  Configuration failed due to an unrecognized configuration option.

- `btLibL2DiscConfigReject`  
  Configuration was rejected (unknown reason).

- `btLibL2DiscConfigUnacceptable`  
  Configuration failed due to unacceptable parameters.

- `btLibL2DiscConnNoResources`  
  The remote device is out of resources.

- `btLibL2DiscConnPsmUnsupported`  
  The remote device does not support the requested protocol service (PSM).

- `btLibL2DiscConnSecurityBlock`  
  The remote device's security system denied the connection.

- `btLibL2DiscLinkDisc`  
  The underlying ACL Link was disconnected.

- `btLibL2DiscQosViolation`  
  The connection was terminated due to a Quality of Service (QOS) violation.

- `btLibL2DiscReasonUnknown`  
  Disconnection occurred for an unknown reason.

- `btLibL2DiscRequestTimeout`  
  An L2CAP request timed out.
**Bluetooth Library: Sockets and Service Discovery**

*Socket Functions*

The Bluetooth library uses sockets to represent L2CAP, RFCOMM, and SDP connections. The functions in this section perform general socket tasks and tasks related to L2CAP and RFCOMM sockets. The functions specific to SDP sockets are in the Service Discovery Protocol Functions section.

---

### New

<table>
<thead>
<tr>
<th></th>
<th>BtLibSocketAdvanceCredit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Purpose</strong></td>
<td>Advance credit to a given RFCOMM connection socket.</td>
</tr>
<tr>
<td><strong>Declared In</strong></td>
<td>BtLib.h</td>
</tr>
<tr>
<td><strong>Prototype</strong></td>
<td>Err BtLibSocketAdvanceCredit (UInt16 btLibRefNum, BtLibSocketRef socket, UInt8 credit)</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td><em>-&gt; btLibRefNum</em> Reference number for the Bluetooth library.</td>
</tr>
<tr>
<td></td>
<td><em>-&gt; socket</em> RFCOMM socket reference number.</td>
</tr>
<tr>
<td></td>
<td><em>-&gt; credit</em> Number credits to add to the total number of credits for this socket. The total number of credits represents the number of packets the remote device can send before data flow stops.</td>
</tr>
<tr>
<td><strong>Result</strong></td>
<td>Returns one of the following values:</td>
</tr>
<tr>
<td></td>
<td><em>btLibErrNoError</em> Success</td>
</tr>
<tr>
<td></td>
<td><em>btLibErrFailed</em> Too many credits advanced.</td>
</tr>
</tbody>
</table>
Bluetooth Library: Sockets and Service Discovery

Socket Functions

btLibErrSocket   The specified socket is invalid.
btlLibErrSocketProtocol   The specified socket is not an RFCOMM socket.
btlLibErrSocketRole   The specified socket is not connected.

Comments

RFCOMM uses a credit based flow control mechanism. For each credit the connection has, one packet of data can be sent. When the credits are spent, data flow stops until you advance more credits using this function.

Multiple calls to this function have a cumulative effect.

New

BtLibSocketClose

Purpose
Close a socket, free associated resources, and kill all associated socket connections.

Declared In
BtLib.h

Prototype
Err BtLibSocketClose (UInt16 btLibRefNum, BtLibSocketRef socket)

Parameters
- btLibRefNum   Reference number for the Bluetooth library.
- socket   Reference number of socket to close.

Result
Returns one of the following values:
- btLibErrNoError   Success.
- btLibErrNotOpen   The referenced Bluetooth library is not open.
- btLibErrSocket   The specified socket is invalid.
btLibErrStackNotOpen
The Bluetooth stack failed to initialize when the library was opened.

Comments
No callback events are generated when closing a socket.

See Also
BtLibSocketCreate, BtLibSocketListen, BtLibSocketConnect, BtLibSocketRespondToConnection

New
BtLibSocketConnect

Purpose
Create an outbound L2CAP or RFCOMM connection.

Declared In
BtLib.h

Prototype
Err BtLibSocketConnect (UInt16 btLibRefNum, BtLibSocketRef socket, BtLibSocketConnectInfoType *connectInfo)

Parameters
- btLibRefNum Reference number for the Bluetooth library.
- socket Reference number of socket to connect.
- connectInfo BtLibSocketConnectInfoType containing Bluetooth device address and protocol-specific connection information.

Result
Returns one of the following values:

btLibErrPending
The results will be returned through a callback event.

btLibErrNoAclLink
An ACL link for the remote device does not exist

btLibErrNotOpen
The referenced Bluetooth library is not open.
btLibErrSocket  The specified socket is invalid.

btLibErrSocketProtocol  The protocol of the specified socket is not supported. This function only supports the L2CAP and RFCOMM protocols.

btLibErrSocketRole  The specified socket is already connected or listening.

btLibErrStackNotOpen  The Bluetooth stack failed to initialize when the library was opened.

Comments  If the connection succeeds, the btLibSocketEventConnectedOutbound event is generated and its status field is set to btLibErrNoError. If connection fails, the same event is generated with a non-zero status field, or a btLibSocketEventDisconnected is generated. In both cases, the status field indicates the reason for the failure.

If the connection succeeds, the btLibSocketEventData event is generated whenever data is received from the remote device. When the channel disconnects, a btLibSocketEventDisconnected event is generated.

BtLibSocketConnectInfoType  The BtLibSocketConnectInfoType structure allows you to specify the address of the remote device and data specific to the protocol of the socket. The protocol-specific data is stored as a union; the member of the union that is valid depends on the protocol.

typedef struct BtLibSocketConnectInfoType {
    BtLibDeviceAddressTypePtr remoteDeviceP;
    union {
        ...
    } data;
} BtLibSocketConnectInfoType;
Field Description

remoteDeviceP A pointer to a `BtLibDeviceAddressType` that contains the address of the remote device.

data A union containing protocol-specific information. This union has two members: `L2Cap`, and `RfComm`.

L2Cap
Use the L2Cap union member if you’re setting up a L2CAP socket. This member contains the following structure:

```c
struct {
    BtLibL2CapPsmType remotePsm;
    UInt16 minRemoteMtu;
    UInt16 localMtu;
} L2Cap;
```

Field Descriptions

remotePsm A `BtLibL2CapPsmType` representing the protocol and service multiplexer (PSM) identifier of the protocol to which this socket should connect. This identifier is obtained using the Service Discovery Protocol (SDP).

minRemoteMtu The minimum MTU, or packet size, that your application can support.

localMtu The MTU, or packet size, of the local device.

RfComm
Use the RfComm union member if you’re setting up a RFCOMM socket. This member contains the following structure:

```c
struct {
    BtLibRfCommServerIdType remoteService;
    UInt16 maxFrameSize;
    UInt8 advancedCredit;
} RfComm;
```

Field Descriptions
remoteService  A `BtLibRfCommServerIdType` representing the RFCOMM service channel on the remote device to which this socket should connect. This identifier is obtained using the Service Discovery Protocol (SDP).

maxFrameSize  The maximum frame size your application can handle. This value must be between `BT_RF_MINFRAMESIZE` and `BT_RF_MAXFRAMESIZE`. If your application can handle any frame size, set this value to `BT_RF_DEFAULT_FRAMESIZE`.

advancedCredit  An amount of credit the socket advances to the remote device when it successfully connects. Additional credit can be advanced using the `BtLibSocketAdvanceCredit` function once the connection has been established.

See Also  `BtLibSocketSend`, `BtLibSocketClose`  

New  `BtLibSocketCreate`

Purpose  Create a socket with foreground notification. The Bluetooth library supports a maximum of 16 socket connections.

Declared In  `BtLib.h`

Prototype  

```c
Err BtLibSocketCreate (UInt16 btLibRefNum,
                        BtLibSocketRef *socketRefP,
                        BtLibSocketProcPtr callbackP, UInt32 refCon,
                        BtLibProtocolEnum socketProtocol)
```

Parameters  

- `-> btLibRefNum`  Reference number for the Bluetooth library.
- `<- socketRefP`  Pointer to an allocated `BtLibSocketRef` that contains the socket value upon return. This pointer must not be `NULL`.  

Socket Functions

-> callbackP    Callback procedure used to respond to socket events. This value must not be NULL.
-> refCon       Caller-defined data to pass to the callback procedure.
-> socketProtocol Protocol (L2CAP, RFCOMM, or SDP) to use with this socket.

Result Returns one of the following values:

btLibErrNoError    Success.
btLibErrNotOpen    The referenced Bluetooth library is not open.
btLibErrParamError Either socketRefP or callbackP is NULL.
btLibErrStackNotOpen The Bluetooth stack failed to initialize when the library was opened.
btLibErrTooMany    The maximum number of sockets allocated for the system has already been reached. The Bluetooth library supports a maximum of 16 socket connections.

Comments No callback events are generated when creating a socket.
Before terminating, applications should close all of the sockets that they have created.

See Also BtLibSocketConnect, BtLibSocketListen, BtLibSocketClose
**BtLibSocketGetInfo**

**Purpose**
Retrieve information for a currently open socket.

**Declared In**
BtLib.h

**Prototype**

```c
Err BtLibSocketGetInfo (UInt16 btLibRefNum, BtLibSocketRef socket, BtLibSocketInfoEnum infoType, void *valueP, UInt32 valueSize)
```

**Parameters**
- `btLibRefNum` Reference number for the Bluetooth library.
- `socket` Reference number for the socket to query.
- `infoType` Type of information to retrieve. See `BtLibSocketInfoEnum`.
- `valueP` Buffer into which this function stores the result. You must allocate this buffer.
- `valueSize` Size, in bytes, of the `valueP` buffer. This size must match the size of the retrieved information.

**Result**
Returns one of the following values:
- `btLibErrNoError` Success.
- `btLibErrNotOpen` The referenced Bluetooth library is not open.
- `btLibErrParamError` One or more parameters is invalid. Be sure that the `valueSize` parameter matches the size of the information you’re retrieving.
Bluetooth Library: Sockets and Service Discovery

Socket Functions

btLibErrSdpNotMapped
The SDP socket has not been mapped to a remote SDP service record. This error occurs when you try to obtain the SDP service record handle before you map socket to a remote service record using BtLibSdpServiceRecordMapRemote.

btLibErrSocket
The specified socket is invalid or not in use.

btLibErrSocketRole
The specified socket is not connected or has the wrong role for the request.

btlibErrSocketProtocol
The specified socket has the wrong protocol for the request.

btLibErrStackNotOpen
The Bluetooth stack failed to initialize when the library was opened.

Comments
BtLibSocketInfoEnum
The BtLibSocketInfoEnum enum allows you to specify which information you want to retrieve using the BtLibSocketGetInfo function.

typedef enum {
    btLibSocketInfo_Protocol = 0,
    btLibSocketInfo_RemoteDeviceAddress,
    btLibSocketInfo_SendPending = 100,
    btLibSocketInfo_MaxTxSize,
    btLibSocketInfo_MaxRxSize,
    btLibSocketInfo_L2CapPsm = 200,
    btLibSocketInfo_L2CapChannel,
    btLibSocketInfo_RfCommServerId = 300,
    btLibSocketInfo_RfCommOutstandingCredits,
    btLibSocketInfo_SdpServiceRecordHandle = 400
} BtLibSocketInfoEnum;
Value Descriptions

btLibSocketInfo_L2CapChannel
BtLibSocketGetInfo returns a BtLibL2CapChannelIDType that represents the channel identifier for this socket. A BtLibL2CapChannelIDType is actually a UInt16. This information is valid for L2CAP sockets only. See the “Logical Link Control and Adaptation Protocol Specification” chapter of the Specification of the Bluetooth System for more information about channel identifiers.

btLibSocketInfo_L2CapPsm
BtLibSocketGetInfo returns a BtLibL2CapPsmType that represents the Protocol and Service Multiplexer (PSM) this socket is using to route packets. This information is only valid for L2CAP sockets.

btLibSocketInfo_MaxRxSize
BtLibSocketGetInfo returns a UInt32 representing the maximum packet size the local device can receive.

btLibSocketInfo_MaxTxSize
BtLibSocketGetInfo returns a UInt32 representing the maximum packet size the local device can transmit.

btLibSocketInfo_Protocol
BtLibSocketGetInfo returns a BtLibProtocolEnum representing the socket’s protocol. The members of this enum are btLibL2CapProtocol, btLibRfCommProtocol, and btLibSdpProtocol.

btLibSocketInfo_RemoteDeviceAddress
BtLibSocketGetInfo returns a BtLibDeviceAddressType representing the address of the device at the other end of this socket.
btLibSocketInfo_RfCommServerId
BtLibSocketGetInfo returns a
BtLibRfCommServerIdType that represents
the socket’s RFCOMM server channel. This
information is valid for RFCOMM sockets only.

btLibSocketInfo_RfCommOutstandingCredits
BtLibSocketGetInfo returns a UInt16
containing the number of remaining credits on
this socket. This information is valid for
RFCOMM sockets only.

btLibSocketInfo_SdpServiceRecordHandle
BtLibSocketGetInfo returns the
BtLibSdpRemoteServiceRecordHandle
for the service record associated with this
socket. This information is valid for SDP
sockets only.

btLibSocketInfo_SendPending
BtLibSocketGetInfo returns a Boolean
indicating whether a send is currently in
progress.

▼

New  BtLibSocketListen

Purpose  Set up an L2CAP or RFCOMM socket as a listener.

Declared In  BtLib.h

Prototype  Err BtLibSocketListen (UInt16 btLibRefNum,
BtLibSocketRef socket,
BtLibSocketListenInfoType *listenInfo)

Parameters  -> btLibRefNum  Reference number for the Bluetooth library.
-> socket  Reference number of the socket.
Bluetooth Library: Sockets and Service Discovery

Socket Functions

<> listenInfo Protocol-specific listening information. For more information see BtLibSocketListenInfoType. This parameter must not be NULL.

Result

Returns one of the following values:

btLibErrNoError Success. The socket is listening for incoming connections.

btLibErrBusy The given PSM is in use (L2CAP only)

btLibErrNotOpen The referenced Bluetooth library is not open.

btLibErrParamError listenInfo is NULL.

btLibErrSocket The specified socket is invalid.

btLibErrSocketProtocol The protocol of the specified socket is not supported. This function only supports the L2CAP and RFCOMM protocols.

btLibErrSocketRole The specified socket is already listening or connected.

btLibErrStackNotOpen The Bluetooth stack failed to initialize when the library was opened.

btLibErrTooMany There are no resources to create a listener socket of this type.

Comments

A listener socket waits for a remote device to initiate a connection to the local device and then generates a btLibSocketEventConnectRequest event to notify the application that it needs to handle the connection attempt.

You need to respond to this event with a call to BtLibSocketRespondToConnection on the listener socket to accept or reject the connection.
Bluetooth Library: Sockets and Service Discovery
Socket Functions

Under certain circumstances, the listenInfo parameter acts as an output as well as an input. See the documentation for BtLibSocketListenInfoType that follows.

**BtLibSocketListenInfoType**
The BtLibSocketListenInfoType structure allows you to specify data specific to the protocol of the listening socket.

```c
typedef struct BtLibSocketListenInfoType {
    union {
        ...
    } data;
} BtLibSocketListenInfoType;
```

This data is stored in the `data` field, which is a union consisting of two members: **L2Cap**, and **RfComm**. The member of the union that is valid depends on the protocol of the listening socket.

**L2Cap**
Use the L2Cap union member if you’re setting up a L2CAP socket as a listener. This member contains the following structure:

```c
struct {
    BtLibL2CapPsmType localPsm;
    UInt16 localMtu;
    UInt16 minRemoteMtu;
} L2Cap;
```

**Field Descriptions**

- **localPsm**
  A `BtLibL2CapPsmType` representing the protocol and service multiplexer (PSM) identifier of the protocol to be used with this socket. You can identify your own protocol provided that its PSM value is odd, is within the range of 0x1001 to 0xFFFF, and has the 9th bit (0x0100) set to zero. These limitations are specified by the *Specification of the Bluetooth System*. If you set this field to `BT_L2CAP_RANDOM_PSM`, the BtLibSocketListen function automatically creates a suitable PSM for the channel and returns it in this structure.
localMtu

The maximum transmission unit (MTU), or packet size, of the local device.

minRemoteMtu

The minimum packet size that your application can support.

**RfComm**

Use the RfComm union member if you’re setting up a RFCOMM socket as a listener. This member contains the following structure:

```
struct {
    BtLibRfCommServerIdType serviceID;
    UInt16 maxFrameSize;
    UInt8 advancedCredit;
} RfComm;
```

**Field Descriptions**

**serviceID**

A `BtLibRfCommServerIdType` representing the socket’s RFCOMM service channel. It is assigned by RFCOMM and returned in this field when you call `BtLibSocketListen`.

**maxFrameSize**

The maximum frame size your application can handle. This value must be between `BT_RF_MINFRAMESIZE` and `BT_RF_MAXFRAMESIZE`. If your application can handle any frame size, set this value to `BT_RF_DEFAULT_FRAMESIZE`.

**advancedCredit**

An amount of credit the socket advances to the remote device when it successfully connects. Additional credit can be advanced using the `BtLibSocketAdvanceCredit` function once the connection has been established.

**See Also**

`BtLibSocketClose`
New

BtLibSocketRespondToConnection

Purpose
Accept or reject an in-bound connection on a given listener socket.

Declared In
BtLib.h

Prototype
Err BtLibSocketRespondToConnection
(UInt16 btLibRefNum, BtLibSocketRef socket, Boolean accept)

Parameters
-> btLibRefNum  Reference number for the Bluetooth library.
-> socket       Reference number of the listener socket.
-> accept       true to accept the connection; false to reject
                the connection.

Result
Returns one of the following values:

btLibErrNoError  Success. This status is returned when accept
                is false.

btLibErrFailed   One or more parameters is invalid.

btLibErrPending  The results will be returned through a callback
                event.

btLibErrNotOpen  The referenced Bluetooth library is not open.

btLibErrSocket   The specified socket is invalid or not in use.

btLibErrSocketProtocol
The protocol of the specified socket is not supported. This function only supports the L2CAP and RFCOMM protocols.

btLibErrSocketRole
The specified socket is not a listener socket.
btLibErrStackNotOpen

The Bluetooth stack failed to initialize when the library was opened.

Comments

You should call this function when you respond to a btLibSocketEventConnectRequest event delivered to a listener socket.

If the connection succeeds, the btLibSocketEventConnectedInbound event is generated and its status field is set to btLibErrNoError. If connection fails, the same event is generated with a non-zero status field, or a btLibSocketEventDisconnected is generated. In both cases, the status field indicates the reason for the failure.

Once the connection succeeds, a btLibSocketEventData callback event is generated whenever data received from the remote device. If the channel disconnects, a btLibSocketEventDisconnected is generated.

RFCOMM listener sockets and L2CAP listener sockets behave differently when you call this function. When you respond to an inbound L2CAP connection, a new L2CAP socket is created to exchange data with the remote device, and the L2CAP listener socket continues to listen for more connections. In other words, a single L2CAP listener socket can “spawn” several L2CAP sockets. This mechanism allows you to create a piconet.

On the other hand, when you respond to an RFCOMM connection, the RFCOMM listener socket becomes a connection socket through which you can exchange data with the remote device. If you want to create another RFCOMM connection, you need to create another listener socket.

See Also

BtLibSocketListen, BtLibSocketSend, BtLibSocketClose
Bluetooth Library: Sockets and Service Discovery

Socket Functions

New

**BtLibSocketSend**

**Purpose**
Send data over a connected L2CAP or RFCOMM socket.

**Declared In**
BtLib.h

**Prototype**
```
Err BtLibSocketSend (UInt16 btLibRefNum,
                     BtLibSocketRef socket, UInt8 *data,
                     UInt32 dataLen)
```

**Parameters**
- `btLibRefNum`  Reference number for the Bluetooth library.
- `socket`  Reference number of the transmitting socket.
- `data`  Pointer to data to send. If the send returns `btLibErrPending`, the data buffer contents must remain intact until the `btLibSocketEventSendComplete` event occurs.
- `dataLen`  Length of data to send. This value must be less than the Maximum Transmission Unit (MTU) for the socket. The MTU indicates the size of the largest packet that the remote device can receive and is determined when the socket is connected.

**Result**
Returns one of the following values:

- `btLibErrPending`  The results will be returned through a callback event.
- `btLibErrBusy`  A send is already in process.
- `btLibErrNoAclLink`  An ACL link for the remote device does not exist.
- `btlibErrNotOpen`  The referenced Bluetooth library is not open.
btLibErrSocket  The specified socket is invalid.

btLibErrSocketProtocol  The protocol of the specified socket is not supported by this function. You can only send using the L2CAP and RFCOMM protocols.

btLibErrSocketRole  The specified socket is not connected.

btLibErrStackNotOpen  The Bluetooth stack failed to initialize when the library was opened.

Comments  When the data has been sent successfully, a btLibSocketEventSendComplete callback event is generated and its status field is set to btLibErrNoError. If the data is not sent successfully, the same callback event is generated with a non-zero status field.

Note that there can be only one send in progress at a time per socket. You must wait for the btLibSocketEventSendComplete event before sending another packet.

See Also  BtLibSocketClose

Service Discovery Protocol Functions

This section describes functions and macros related to the Bluetooth Service Discovery Protocol (SDP).
New BtLibSdpCompareUuids

Purpose
Compare two UUIDs.

Declared In
BtLib.h

Prototype
Err BtLibSdpCompareUuids (UInt16 btLibRefNum, BtLibSdpUuidType *uuid1, BtLibSdpUuidType *uuid2)

Parameters
- btLibRefNum Reference number for the Bluetooth Library.
- uuid1 UUID to compare.
- uuid2 UUID to compare.

Result
Returns one of the following values:
btLibErrNoError
UUIDs are the same
btLibErrError
UUIDs are different.
btLibErrNotOpen
The referenced Bluetooth library is not open.
btLibErrParamError
One or both UUIDs are invalid.
btLibErrStackNotOpen
The Bluetooth stack failed to initialize when the library was opened.
**New**  

**BtLibSdpGetPSMByUuid**

**Purpose**  
Get an available L2CAP PSM using SDP.

**Declared In**  
BtLib.h

**Prototype**  
Err BtLibSdpGetPsmByUuid (UInt16 btLibRefNum,  
BtLibSocketRef socket,  
BtLibDeviceAddressType *remoteDeviceP,  
BtLibSdpUuidType *serviceUuidList,  
UInt8 uuidListLen)

**Parameters**  
- btLibRefNum Reference number for the Bluetooth library.
- socket Reference number for an SDP socket.
- remoteDeviceP Device address of a remote device to query. This parameter must not be NULL.
- serviceUuidList Array of UUIDs that must match those of the service record. This parameter must not be NULL.
- uuidListLen Length of serviceUuidList. A maximum of 12 entries is allowed.

**Result**  
Returns one of the following values:

- btLibErrPending  
The PSM value will be returned through a callback event.

- btLibErrNotOpen  
The referenced Bluetooth library is not open.

- btLibErrOutOfMemory  
Not enough memory to complete request

- btLibErrParamError  
One or more parameters is invalid.
btLibErrSocket The specified socket is invalid or not in use.

btLibErrSocketRole The specified socket is not connected.

btLibErrStackNotOpen The Bluetooth stack failed to initialize when the library was opened.

Comments This function returns the L2CAP PSM of the first SDP record on the remote device that contains all the specified UUIDs.

This function generates a btLibSocketEventSdpGetPsmByUuid event when the query completes or fails.

See Also BtLibSdpGetServerChannelByUuid

New BtLibSdpGetRawDataElementSize

Purpose Macro that returns a constant representing the data element’s size.

Declared In BtLib.h

Prototype BtLibSdpGetRawDataElementSize (header)

Parameters -> header First byte of a data element

Result A constant representing the size of the data element.

Comments The first byte of a SDP data element contains the type and size of the data element.

The result of this macro is one of the following constants:

Data Element Sizes

btLibDESD_1BYTE A 1-byte element. However, if the data element’s type is btLibDETD_NIL then the size is 0 bytes.
btLibDESD_2BYTES
   A 2-byte element.
btLibDESD_4BYTES
   A 4-byte element.
btLibDESD_8BYTES
   An 8-byte element.
btLibDESD_16BYTES
   A 16-byte element.
btLibDESD_ADD_8BITS
   The element's actual data size, in bytes, is contained in the next 8 bits.
btLibDESD_ADD_16BITS
   The element's actual data size, in bytes, is contained in the next 16 bits.
btLibDESD_ADD_32BITS
   The element's actual data size, in bytes, is contained in the next 32 bits.

These size constants are discussed in greater detail in the “Service Discovery Protocol” chapter of the Specification of the Bluetooth System.

See Also
BtLibSdpGetRawDataElementType,
BtLibSdpParseRawDataElement,
BtLibSdpVerifyRawDataElement
New  BtLibSdpGetRawDataElementType

Purpose  Macro that returns an SDP data element’s type.

Declared In  BtLib.h

Prototype  BtLibSdpGetRawDataElementType (header)

-> header  The first byte of a data element

Result  The type of the data element.

Comments  The first byte of a SDP data element contains the type and size of the data element.

The result of this macro is one of the following constants:

Data Element Types

btLibDETD_NIL  Nil, the null type
btLibDETD_UINT  Unsigned Integer.
btLibDETD_SINT  Signed Integer
btLibDETD_UUID  UUID, a universally unique identifier
btLibDETD_TEXT  Text string
btLibDETD_BOOL  Boolean
btLibDETD_SEQ  Data element sequence
btLibDETD_ALT  Data element alternative
btLibDETD_URL  URL, a uniform resource locator

These types are discussed in greater detail in the “Service Discovery Protocol” chapter of the Specification of the Bluetooth System.

See Also  BtLibSdpGetRawDataElementSize, BtLibSdpParseRawDataElement, BtLibSdpVerifyRawDataElement
New

**BtLibSdpGetServerChannelByUuid**

**Purpose**  
Get an available RFCOMM server channel using SDP.

**Declared In**  
BtLib.h

**Prototype**  
```
Err BtLibSdpGetServerChannelByUuid
(UInt16 btLibRefNum, BtLibSocketRef socket, 
BtLibDeviceAddressType *remoteDeviceP, 
BtLibSdpUuidType *serviceUuidList, 
UInt8 uuidListLen)
```

**Parameters**  
- `btLibRefNum` Reference number for the Bluetooth library.
- `socket` Reference number for an SDP socket.
- `remoteDeviceP` Device address of a remote device to query. This parameter must not be NULL.
- `serviceUuidList` Array of UUIDs that must match those of the service record. This parameter must not be NULL.
- `uuidListLen` Length of `serviceUuidList`. A maximum of 12 entries is allowed.

**Result**  
Returns one of the following values:

- `btLibErrPending`  
The server channel will be returned through a callback event.

- `btLibErrNotOpen`  
The referenced Bluetooth library is not open.

- `btLibErrOutOfMemory`  
Not enough memory to complete request

- `btLibErrParamError`  
One or more parameters is invalid.
Bluetooth Library: Sockets and Service Discovery

Service Discovery Protocol Functions

btLibErrSocket  The specified socket is invalid or not in use.
bLibErrSocketRole  The specified socket is not connected.
bLibErrStackNotOpen  The Bluetooth stack failed to initialize when the library was opened.

Comments  This function returns the RFCOMM server channel number of the first SDP record on the remote device that contains all the specified UUIDs.

This function generates a btLibSocketEventSdpGetServerChannelBy_Uuid event when the query completes or fails.

See Also  BtLibSdpGetPSMByUuid

New  BtLibSdpParseRawDataElement

Purpose  Parse a raw SDP data element to determine where the data field begins and the size of the data field.

Declared In  BtLib.h

Prototype  Err BtLibSdpParseRawDataElement
          (UInt16 btLibRefNum, const UInt8 *dataElementP,
           UInt16 *offset, UInt32 *length)

Parameters  -> btLibRefNum  Reference number for the Bluetooth library.

            -> dataElementP  Pointer to a raw SDP data element.

            <- offset  Offset, in bytes, between dataElementP and the start of the data field.
<- lengthP Length, in bytes, of the data field.

**Result**

Returns one of the following values:

- **btLibErrNoError**
  Successfully parsed the attribute.

- **btLibErrNotOpen**
  The reference Bluetooth library is not open.

- **btLibErrParamError**
  `dataElementP`, `offset`, or `length` is NULL.

- **btLibErrStackNotOpen**
  The Bluetooth stack failed to initialize when the library was opened.

**Comments**

A data element has three fields. The first field, called the **header field**, identifies the type of value stored in the data element and the size of the element. The second field, called the **size field**, contains more information about the size of the data if it’s not completely specified by the header. Otherwise the size field is omitted. The third field, called the **data field**, contains the data element’s actual value.

The offset this function returns is the offset between the start of the data element and the data field. The size this function returns is the size of the data field. Note that the sum of the offset and the size is the size of the data element.

This function is especially useful for iterating through entries in a list attribute.

The *Specification of the Bluetooth System* has more information about the structure of a data element.

**See Also**

- `BtLibSdpVerifyRawDataElement`
- `BtLibSdpGetRawDataElementType`
- `BtLibSdpGetRawDataElementSize`
New

**BtLibSdpServiceRecordCreate**

**Purpose**
Allocate a memory chunk that represents an SDP service record.

**Declared In**
BtLib.h

**Prototype**
```
Err BtLibSdpServiceRecordCreate
    (UInt16 btLibRefNum,
     BtLibSdpRecordHandle *recordH)
```

**Parameters**
- `btLibRefNum` Reference number for the Bluetooth library.
- `recordH` SDP memory handle for the new SDP memory record.

**Result**
Returns one of the following values:
- `btLibErrNoError` Success.
- `btLibErrNotOpen` The referenced Bluetooth library is not open.
- `btLibErrOutOfMemory` Not enough memory to allocate the memory chunk.
- `btLibErrParamError` `recordH` is NULL.
- `btLibErrStackNotOpen` The Bluetooth stack failed to initialize when the library was opened.

**See Also**
- BtLibSdpServiceRecordDestroy
- BtLibSdpServiceRecordStartAdvertising
- BtLibSdpServiceRecordStopAdvertising
**New**

### BtLibSdpServiceRecordDestroy

**Purpose** Free the memory associated with a SDP memory record.

**Declared In** BtLib.h

**Prototype**

```c
Err BtLibSdpServiceRecordDestroy
(UINT16 btLibRefNum,
BtLibSdpRecordHandle recordH)
```

**Parameters**

- `btLibRefNum` Reference number for the Bluetooth library.
- `recordH` SDP memory handle associated with the memory chunk to be freed.

**Result** Returns one of the following values:

- `btLibErrNoError` Success.
- `btLibErrNotOpen` The referenced Bluetooth library is not open.
- `btLibErrParamError` `recordH` does not refer to a valid SDP memory record.
- `btLibErrStackNotOpen` The Bluetooth stack failed to initialize when the library was opened.

**Comments** This function stops advertising the record before it frees it.

**See Also**

- `BtLibSdpServiceRecordCreate`
- `BtLibSdpServiceRecordStartAdvertising`
- `BtLibSdpServiceRecordStopAdvertising`
New

BtLibSdpServiceRecordGetAttribute

Purpose
Retrieve the value of a specific attribute in a SDP memory record. If the attribute is a list or a protocol descriptor list (a list of lists), this function retrieves the value of a specific list entry.

Declared In
BtLib.h

Prototype
Err BtLibSdpServiceRecordGetAttribute
(UInt16 btLibRefNum,
BtLibSdpRecordHandle recordH,
BtLibSdpAttributeIdType attributeID,
BtLibSdpAttributeDataType *attributeValues,
UInt16 listNumber, UInt16 listEntry)

Parameters
- btLibRefNum Reference number for the Bluetooth library.
- recordH Handle identifying the SDP memory record.
- attributeID Attribute identifier of the attribute to retrieve.

attributeValues
Buffer into which this function stores the attribute's value. You must allocate this buffer. This pointer must not be NULL.

- listNumber List to query if the attribute is a protocol descriptor list. Otherwise this parameter is ignored.

- listEntry Item to get in the list if the attribute is a list attribute. Otherwise this parameter is ignored.

Result
Returns one of the following values:

btLibErrNoError
Success.

btLibErrPending
The specified SDP memory record refers to a service record on a remote device. The result will be returned through a callback event.
btLibErrBusy

The connection is parked. This error can occur only if the SDP memory record refers to a service record on a remote device.

btLibErrInProgress

A query is already pending on this socket. This error can occur only if the SDP memory record refers to a service record on a remote device.

btLibErrNoAclLink

An ACL link to the remote device does not exist.

btLibErrNotOpen

The referenced Bluetooth library is not open.

btLibErrOutOfRangeMemory

Not enough memory to perform the query.

btLibErrParamError

recordH is an invalid handle or attributeValues is NULL.

btLibErrSdpAttributeNotSet

The specified attribute does not exist in the specified service record.

btLibErrStackNotOpen

The Bluetooth stack failed to initialize when the library was opened.

Comments

If the specified SDP memory record refers to a service record on a remote device, this function generates a btLibSocketEventSdpGetAttribute event when the result is available or the query fails. In this case, the buffer to which attributeValues points must not be freed before the event occurs; making the buffer global ensures that it remains over the duration of the SDP query.

If you are retrieving a string or a URL, you need to allocate additional space. See the documentation for BtLibSdpAttributeDataType for more information.
This function supports the universal attributes defined in “Service Discovery Protocol” chapter of the Specification of the Bluetooth System.

See Also
BtLibSdpServiceRecordSetAttribute,
BtLibSdpServiceRecordMapRemote,
BtLibSdpServiceRecordGetNumListEntries,
BtLibSdpServiceRecordGetNumLists,
BtLibSdpServiceRecordGetStringOrUrlLength

New
BtLibSdpServiceRecordGetNumListEntries

Purpose
Get the number of entries in a list attribute.

Declared In
BtLib.h

Prototype
Err BtLibSdpServiceRecordGetNumListEntries
(UInt16 btLibRefNum,
BtLibSdpRecordHandle recordH,
BtLibSdpAttributeIdType attributeID,
UInt16 listNumber, UInt16 *numEntries)

Parameters
-> btLibRefNum Reference number for the Bluetooth library.
-> recordH Handle identifying the SDP memory record.
-> attributeID Attribute identifier of the attribute whose number of list entries is retrieved.
-> listNumber List to query if the attribute is a ProfileDescriptorListEntry. Otherwise this parameter is ignored.
<- numEntries Number of entries in the list.

Result
Returns one of the following values:
bLibErrNoError Success
btLibErrPending
The specified SDP memory record refers to a service record on a remote device. The result will be returned through a callback event.

btLibErrBusy
The connection is parked. This error can occur only if the SDP memory record refers to a service record on a remote device.

btLibErrInProgress
Another query is pending on this socket. This error can occur only if the SDP memory record refers to a service record on a remote device.

btLibErrNoAclLink
An ACL link to the remote device does not exist.

btLibErrNotOpen
The referenced Bluetooth library is not open.

btLibErrOutOfMemory
Not enough memory to perform this query.

btLibErrParamError
recordH is an invalid handle or numEntries is NULL.

btLibErrSdpAttributeNotSet
The specified attribute does not exist in the specified service record.

btLibErrStackNotOpen
The Bluetooth stack failed to open when the library was opened.

Comments
This function supports the universal attributes defined in “Service Discovery Protocol” chapter of the Specification of the Bluetooth System. Specifically, this function gives valid results for ServiceClassIdList, ProtocolDescriptorList, BrowseGroupList, LanguageBaseAttributeIDList, and ProfileDescriptorList attributes.

If the specified SDP memory record refers to a service record on a remote device, this function generates a
**Bluetooth Library: Sockets and Service Discovery**

*Service Discovery Protocol Functions*

btLibSocketEventSdpGetNumListEntries event when the result is available or the query fails.

**See Also**

BtLibSdpServiceRecordGetNumLists,
BtLibSdpServiceRecordGetAttribute,
BtLibSdpServiceRecordGetStringOrUrlLength,
BtLibSdpServiceRecordMapRemote

---

**New**

<table>
<thead>
<tr>
<th>BtLibSdpServiceRecordGetNumLists</th>
</tr>
</thead>
</table>

**Purpose**

Get the number of lists in a protocol descriptor list SDP attribute.

**Declared In**

BtLib.h

**Prototype**

Err BtLibSdpServiceRecordGetNumLists
(UInt16 btLibRefNum,
BtLibSdpRecordHandle recordH,
BtLibSdpAttributeIdType attributeID,
UInt16 *numLists)

**Parameters**

- `btLibRefNum` Reference number for the Bluetooth library.
- `recordH` Handle identifying the SDP memory record.
- `attributeID` Attribute identifier of the attribute whose number of lists is retrieved.
- `numLists` Number of lists.

**Result**

Returns one of the following values:

- `btLibErrNoError` Success.
- `btLibErrPending` The specified SDP memory record refers to a service record on a remote device. The result will be returned through a callback event.
btLibErrBusy
The connection is parked. This error can occur only if the SDP memory record refers to a service record on a remote device.

btLibErrInProgress
Another query is pending on this socket. This error can occur only if the SDP memory record refers to a service record on a remote device.

btLibErrNoAclLink
An ACL link to the remote device does not exist.

btLibErrNotOpen
The referenced Bluetooth library is not open.

btLibErrOutOfMemory
Not enough memory to perform this query.

btLibErrParamError
recordH is an invalid handle or numLists is NULL.

btLibErrSdpAttributeNotSet
The specified attribute does not exist in the specified service record.

btLibErrStackNotOpen
The Bluetooth stack failed to open when the library was opened.

Comments
If the specified SDP memory record refers to a service record on a remote device, this function generates a btLibSocketEventSdpGetNumLists event when the result is available or the query fails.

See Also
BtLibSdpServiceRecordGetNumListEntries, BtLibSdpServiceRecordGetAttribute, BtLibSdpServiceRecordGetStringOrUrlLength, BtLibSdpServiceRecordMapRemote
New **BtLibSdpServiceRecordGetRawAttribute**

**Purpose** Retrieve the value of an attribute of an SDP memory record. The retrieved attribute is in the format defined in the “Service Discovery Protocol” chapter of the *Specification of the Bluetooth System*.

**Declared In** BtLib.h

**Prototype**
```
Err BtLibSdpServiceRecordGetRawAttribute
(UInt16 btLibRefNum,
 BtLibSdpRecordHandle recordH,
 BtLibSdpAttributeIdType attributeID,
 UInt8 *value, UInt16 *valSize)
```

**Parameters**
- **-> btLibRefNum** Reference number for the Bluetooth library.
- **-> recordH** Handle identifying the SDP memory record.
- **-> attributeID** Attribute identifier of the attribute to retrieve.
- **<- value** Buffer into which this function stores the retrieved SDP attribute data. You must allocate this buffer. This pointer must not be `NULL`.
- **<- valSize** Size of the `value` buffer upon entry. This parameter must not be zero. Upon return, contains the number of bytes retrieved.

**Result** Returns one of the following values:

- **btLibErrNoError** Success.
- **btLibErrPending** The specified SDP memory record refers to a service record on a remote device. The result will be returned through a callback event.
- **btLibErrBusy** The connection is parked. This error can occur only if the SDP memory record refers to a service record on a remote device.
btLibErrInProgress
A query is already pending on this socket. This error can occur only if the SDP memory record refers to a service record on a remote device.

btLibErrNoAclLink
An ACL link to the remote device does not exist.

btLibErrNotOpen
The referenced Bluetooth library is not open.

btLibErrOutOfMemory
Not enough memory to perform the query.

btLibErrParamError
recordH is an invalid handle, value is NULL, valSize is 0, or the size of the attribute value is larger than valSize.

btLibErrSdpAttributeNotSet
The specified attribute does not exist in the specified service record.

btLibErrStackNotOpen
The Bluetooth stack failed to initialize when the library was opened.

Comments
If the specified SDP memory record refers to a service record on a remote device, this function generates a btLibSocketEventSdpGetRawAttribute event when the result is available or the query fails.

See Also
BtLibSdpServiceRecordSetRawAttribute, BtLibSdpServiceRecordGetSizeOfRaw Attribute, BtLibSdpServiceRecordMapRemote
New BtLibSdpServiceRecordGetSizeOfRawAttribute

Purpose
Return the size, in bytes, of any attribute of an SDP memory record.

Declared In
BtLib.h

Prototype
Err BtLibSdpServiceRecordGetSizeOfRawAttribute
(UInt16 btLibRefNum,
BtLibSdpRecordHandle recordH,
BtLibSdpAttributeIdType attributeID,
UInt16 *size)

Parameters
-> btLibRefNum Reference number for the Bluetooth library.
-> recordH Handle identifying the SDP memory record.
-> attributeID Attribute identifier of the attribute whose size
   is retrieved.
<- size Pointer to a UInt16 to store the size of the attribute. This parameter must not be NULL.

Result
Returns one of the following values:
btLibErrNoError Success.
btLibErrPending The specified SDP memory record refers to a service record on a remote device. The result will be returned through a callback event.
btLibErrBusy The connection is parked. This error can occur only if the SDP memory record refers to a service record on a remote device.
btLibErrInProgress
A query is already pending on this socket. This error can occur only if the SDP memory record refers to a service record on a remote device.

btLibErrNoAclLink
An ACL link to the remote device does not exist.

btLibErrNotOpen
The referenced Bluetooth library is not open.

btLibErrOutOfMemory
Not enough memory to perform the query.

btLibErrParamError
recordH is an invalid handle or size is NULL.

btLibErrSdpAttributeNotSet
The specified attribute does not exist in the specified service record.

btLibErrStackNotOpen
The Bluetooth stack failed to initialize when the library was opened.

Comments
If the specified SDP memory record refers to a service record on a remote device, this function generates a btLibSocketEventSdpGetRawAttributeSize event when the result is available or the query fails.

See Also
BuLibSdpServiceRecordGetRawAttribute, BuLibSdpServiceRecordMapRemote, BuLibSdpServiceRecordSetRawAttribute
New BtLibSdpServiceRecordGetStringOrUrlLength

Purpose Get the length of a string or URL attribute in a SDP memory record.

Declared In BtLib.h

Prototype Err BtLibSdpServiceRecordGetStringOrUrlLength (UInt16 btLibRefNum, BtLibSdpRecordHandle recordH, BtLibSdpAttributeIdType attributeID, UInt16 *length)

Parameters

- -> btLibRefNum Reference number for the Bluetooth library.
- -> recordH Handle identifying the SDP memory record.
- -> attributeID Attribute identifier of the attribute whose length is retrieved.
- <- length Pointer to a UInt16 where the length of the attribute is stored. This parameter cannot be NULL.

Result Returns one of the following values:

- btLibErrNoError Success.
- btLibErrPending The specified SDP memory record refers to a service record on a remote device. The result will be returned through a callback event.
- btLibErrBusy The connection is parked. This error can occur only if the SDP memory record refers to a service record on a remote device.
btLibErrInProgress
A query is already pending on this socket. This error can occur only if the SDP memory record refers to a service record on a remote device.

btLibErrNoAclLink
An ACL link to the remote device does not exist.

btLibErrNotOpen
The referenced Bluetooth library is not open.

btLibErrOutOfMemory
Not enough memory to perform the query.

btLibErrParamError
The recordH does not refer to a valid handle, length is NULL, or the attribute is not a string or a URL.

btLibErrSdpAttributeNotSet
The specified attribute does not exist in the specified SDP record.

btLibErrStackNotOpen
The Bluetooth stack failed to initialize when the library was opened.

Comments
Bluetooth strings do not include a null terminator.
If the SDP memory record refers to a service record on a remote device, this function generates a
btLibSocketEventSdpGetStringLen event when the result is available or the query fails.

See Also
BtLibSdpServiceRecordGetAttribute,
BtLibSdpServiceRecordGetNumListEntries,
BtLibSdpServiceRecordGetNumLists,
BtLibSdpServiceRecordMapRemote
**New**  
**BtLibSdpServiceRecordMapRemote**

**Purpose**  
Configure an SDP memory record so it refers to a service record on a remote device.

**Declared In**  
BtLib.h

**Prototype**  
```c
Err BtLibSdpServiceRecordMapRemote
(UInt16 btLibRefNum, BtLibSocketRef socket,
 BtLibDeviceAddressType *remoteDeviceP,
 BtLibSdpRemoteServiceRecordHandle remoteHandle,
 BtLibSdpRecordHandle recordH)
```

**Parameters**
- `-> btLibRefNum`  
  Reference number for the Bluetooth library.
- `-> socket`  
  Reference number of an SDP socket.
- `-> remoteDeviceP`  
  Device to query.
- `-> remoteHandle`  
  Remote service record handle.
- `-> recordH`  
  SDP memory handle of an empty SDP record.

**Result**  
Returns one of the following values:
- `btLibErrNoError`  
  The mapping was successful.
- `btLibErrNotOpen`  
  The referenced Bluetooth library is not open.
- `btLibErrOutOfMemory`  
  Not enough memory to perform mapping.
- `btLibErrParamError`  
  `recordH` is invalid or refers to an invalid memory chunk.
- `btLibErrSdpMapped`  
  The SDP memory record is already mapped to a remote service record.
btLibErrSocket  The specified socket is invalid or not in use.

btLibErrSocketProtocol  The specified socket is not an SDP socket.

btLibErrStackNotOpen  The Bluetooth stack failed to initialize when the library was opened.

Comments  You must create an SDP memory record using BtLibSdpServiceRecordCreate before using this function. Note that this function does not copy the contents of the remote service record to the SDP memory record in local memory.

New  BtLibSdpServiceRecordSetAttribute

Purpose  Set the value of an attribute in an SDP memory record. If the attribute is a list or a protocol descriptor list (a list of lists), this function sets the value of a specific list entry. The SDP memory record must represent a local unadvertised service record.

Declared In  BtLib.h

Prototype  Err BtLibSdpServiceRecordSetAttribute
(UInt16 btLibRefNum, BtLibSdpRecordHandle recordH, BtLibSdpAttributeIdType attributeID, BtLibSdpAttributeDataType *attributeValue, UInt16 listNumber, UInt16 listEntry)

Parameters  -> btLibRefNum  Reference number for the Bluetooth library.
        -> recordH  Handle of the service record to modify.
        -> attributeID  Attribute identifier of the attribute to set.
        -> attributeValue  Pointer to the new value for the attribute. This pointer must not be NULL.
Bluetooth Library: Sockets and Service Discovery

Service Discovery Protocol Functions

-> listNumber  List to modify if the attribute is a protocol descriptor list. Otherwise this parameter is ignored.

-> listEntry  Item to set in the list if the attribute is a list attribute. Otherwise this parameter is ignored.

Result  Returns one of the following values:

btLibErrNoError  Success.

btLibErrAdvertised  An advertised record was passed in recordH. The record must not be advertised.

btLibErrNotOpen  The referenced Bluetooth library is not open.

btLibErrOutOfMemory  Not enough memory to set the attribute.

btLibErrParamError  recordH is invalid or attributeValue is NULL.

btLibErrRemoteRecord  A remote record was passed in recordH. The record must be local.

btLibErrStackNotOpen  The Bluetooth stack failed to initialize when the library was opened.

Comments  This function only works on SDP memory records that are local and not advertised. You can advertise the record after you finish modifying it.

This function supports the universal attributes defined in the Specification of the Bluetooth System.

See Also  BtLibSdpServiceRecordGetAttribute, BtLibSdpServiceRecordStartAdvertising, BtLibSdpServiceRecordStopAdvertising
**New** \textbf{BtLibSdpServiceRecordSetAttributesForSocket}

**Purpose** Initialize an SDP memory record so it can represent an existing L2CAP or RFCOMM listener socket as a service.

**Declared In** BtLib.h

**Prototype**

\begin{verbatim}
Err BtLibSdpServiceRecordSetAttributesForSocket
(UInt16 btLibRefNum, BtLibSocketRef socket,
BtLibSdpUuidType *serviceUUIDList,
UInt8 uuidListLen, const Char *serviceName,
UInt16 serviceNameLen,
BtLibSdpRecordHandle recordH)
\end{verbatim}

**Parameters**

- `btLibRefNum` Reference number for the Bluetooth library.
- `socket` Reference number for an RFCOMM or L2CAP socket in listening mode.
- `serviceUUIDList` List of UUIDs for the service record.
- `uuidListLen` Number of entries in `serviceUUIDList`. A maximum of 12 entries is allowed.
- `serviceName` User-friendly name for the service in English.
- `serviceNameLen` Size, in bytes, of `serviceName`.
- `recordH` Handle of the service record to be initialized.

**Result** Returns one of the following values:

- `btLibErrNoError` Success.
- `btLibErrAdvertised` The record specified by `recordH` is being advertised. You must stop advertising the record before you can change it.
btLibErrNotOpen
The referenced Bluetooth library is not open.

btLibErrOutOfMemory
Not enough memory to store the contents of the SDP record.

btLibErrParamError
recordH is not a valid record handle.

btLibErrRemoteRecord
A remote record was passed in recordH. Because the service is local, the record must be local.

btLibErrSocket
The specified socket is invalid or not in use.

btLibErrSocketRole
The specified socket is not a listener socket.

btLibErrStackNotOpen
The Bluetooth stack failed to initialize when the library was opened.

Comments
You must first create an SDP record using BtLibSdpServiceRecordCreate. However, the record must not be advertised. In other words, don’t call BtLibSdpServiceRecordStartAdvertising until after calling this function.

See Also
BtLibSdpServiceRecordCreate, BtLibSocketListen

New
BtLibSdpServiceRecordSetRawAttribute

Purpose
Set the value for an attribute of a SDP memory record. This function allows you to specify the attribute as an array of bytes in the format defined in the “Service Discovery Protocol” chapter of the
**Specification of the Bluetooth System.** The SDP memory record must represent a local unadvertised service record.

**Declared In**

BtLib.h

**Prototype**

```c
Err BtLibSdpServiceRecordSetRawAttribute
(UInt16 btLibRefNum,
BtLibSdpRecordHandle recordH,
BtLibSdpAttributeIdType attributeID,
const Uint8 *value, Uint16 valSize)
```

**Parameters**

- `btLibRefNum`  Reference number for the Bluetooth library.
- `recordH`  Handle identifying the SDP memory record.
- `attributeID`  Attribute identifier of the attribute to set.
- `value`  Array of bytes containing SDP attribute data in the format defined in the SDP protocol. This parameter must not be **NULL**.
- `valSize`  Size, in bytes, of `value`. This parameter must not be 0.

**Result**  Returns one of the following values:

- `btLibErrNoError`  Success.
- `btLibErrAdvertised`  `recordH` is being advertised. The record must not be advertised.
- `btLibErrNotOpen`  The referenced Bluetooth library is not open.
- `btLibErrOutOfMemory`  Not enough memory to set the attribute.
- `btLibErrParamError`  `recordH` is invalid, `value` is NULL, or `valSize` is 0.
Bluetooth Library: Sockets and Service Discovery

Service Discovery Protocol Functions

btLibErrRemoteRecord

recordH refers to a service record on a remote device. The service record must be local.

btLibErrStackNotOpen

The Bluetooth stack failed to initialize when the library was opened.

Comments

If the service record is being advertised, you must stop advertising it before you modify it.

See Also

BtLibSdpServiceRecordGetRawAttribute, BtLibSdpServiceRecordSetAttribute, BtLibSdpServiceRecordStopAdvertising, BtLibSdpServiceRecordStartAdvertising

New

BtLibSdpServiceRecordsGetByServiceClass

Purpose

Get the service record handles corresponding to the service classes advertised on a remote device.

Declared In

BtLib.h

Prototype

Err BtLibSdpServiceRecordsGetByServiceClass (UInt16 btLibRefNum, BtLibSocketRef socket, BtLibDeviceAddressType *remoteDeviceP, BtLibSdpUuidType *uuidList, UInt16 uuidListLen, BtLibSdpRemoteServiceRecordHandle *srvRecList, UInt16 *numSrvRec)

Parameters

- > btLibRefNum  Reference number for the Bluetooth library.
- > socket  Reference number of an SDP socket.
- > remoteDevice  Remote device to query.
Bluetooth Library: Sockets and Service Discovery

Service Discovery Protocol Functions

- `uuidList` Array of UUIDs identifying the service classes. This parameter must not be NULL.

- `uuidListLen` Number of elements in the `uuidList`. You can specify a maximum of 12 UUIDs.

- `<srvRecList` Array of service record handles into which this function stores the results of the SDP query. You must allocate this array. This pointer must not be NULL.

- `numSrvRec` Number of service records allocated in `srvRecList`. This value is sent to the SDP server so it can limit the number of responses. On return, the actual number of records retrieved.

Result

Returns one of the following values:

- `btLibErrPending` The results will be returned through a callback event.

- `btLibErrBusy` The connection to the remote device is parked.

- `btLibErrInProgress` A SDP query is already in progress on this socket.

- `btLibErrNoAclLink` An ACL link to the remote device does not exist.

- `btLibErrNotOpen` The referenced Bluetooth library is not open.

- `btLibErrOutOfMemory` Not enough memory to perform the query.

- `btLibErrParamError` One or more parameters are invalid.

- `btLibErrSocket` The specified socket is invalid or not in use.

- `btLibErrSocketProtocol` The specified socket is not an SDP socket.
btLibErrStackNotOpen
The Bluetooth stack failed to initialize when the library was opened.

Comments
You need to allocate srvRecList, an array of BtLibSdpRemoteServiceRecordHandles large enough to accommodate all of the service record handles corresponding to the specified service classes. Specify the size of the array using the numSrvRec parameter.

This function generates a btLibSocketEventSdpServiceRecordHandle event when the matching service records are available or the query fails.

New BtLibSdpServiceRecordStartAdvertising

Purpose
Make visible an SDP memory record representing a local SDP service record. Remote devices can access visible service records through SDP.

Declared In BtLib.h

Prototype
Err BtLibSdpServiceRecordStartAdvertising
(UInt16 btLibRefNum,
BtLibSdpRecordHandle recordH)

Parameters
- -> btLibRefNum Reference number for the Bluetooth library.
- -> recordH Handle of the service record to make available to remote devices.

Result
Returns one of the following values:
btLibErrNoError
Success
btLibErrNotOpen
The referenced Bluetooth library is not open.
btLibErrParamError
  recordH is not a valid record handle.

btLibErrRemoteRecord
  recordH refers to a remote record. The record
  must be local.

btLibErrSdpAdvertised
  The service record is already accessible by
  remote devices.

btLibErrStackNotOpen
  The Bluetooth stack failed to initialize when the
  library was opened.

Comments
You cannot modify an SDP memory record while it is available to
remote devices.

See Also BtLibSdpServiceRecordStopAdvertising

New BtLibSdpServiceRecordStopAdvertising

Purpose
Hide an SDP memory record representing a local SDP service
record. Remote devices cannot access hidden service records
through SDP.

Declared In BtLib.h

Prototype
Err BtLibSdpServiceRecordStopAdvertising
  (UInt16 btLibRefNum,
   BtLibSdpRecordHandle recordH)

Parameters
-> btLibRefNum  Reference number for the Bluetooth library.
-> recordH      Handle of the service record to hide.

Result
Returns one of the following values:
btLibErrNoError  
Success. The SDP record is no longer available to remote devices.

btLibErrNotOpen  
The referenced Bluetooth library is not open.

btLibErrParamError  
recordH is not a valid record handle.

btLibErrRemoteRecord  
recordH refers to a remote record. The record must be local.

btLibErrSdpNotAdvertised  
The service record is already hidden from remote devices.

btLibErrStackNotOpen  
The Bluetooth stack failed to initialize when the library was opened.

See Also  
BtLibSdpServiceRecordStartAdvertising

\[
\textbf{New} \quad \textbf{BtLibSdpUuidInitialize}
\]

Purpose  
Macro that sets the value of a UUID.

Declared In  
BtLib.h

Prototype  
BtLibSdpUuidInitialize (uuid, value, valSize)

Parameters  
- **uuid**  
  BtLibSdpUuidType to initialize.
- **value**  
  Array of bytes representing the UUID. The size of this array depends on valSize.
- **valSize**  
  BtLibSdpUuidSizeEnum member specifying the size of the value array.

Result  
None.
**New**

**BtLibSdpVerifyRawDataElement**

**Purpose**
Verify that a raw SDP data element is properly formed.

**Declared In**
BtLib.h

**Prototype**
```c
Err BtLibSdpVerifyRawDataElement
    (UInt16 btLibRefNum, const UInt8 *value,
    UInt16 valSize, UInt8 maxLevel)
```

**Parameters**
- `btLibRefNum` Reference number for the Bluetooth library.
- `value` Raw SDP attribute data.
- `valSize` Size of `value`, in bytes. The size of the data element must be less than or equal to this parameter, otherwise this function fails.
- `maxLevel` Maximum level of recursion over which this function verifies the data element. Must be at least one.

**Result**
Returns one of the following values:
- `btLibErrNoError` SDP data element is properly formatted.
- `btLibErrError` SDP data element is not properly formatted.
- `btLibErrNotOpen` The reference Bluetooth library is not open.
- `btLibErrParamError` `value` is NULL.
- `btLibErrStackNotOpen` The Bluetooth stack failed to initialize when the library was opened.

**Comments**
This function checks all size descriptors in the element to ensure that the data element fits into the indicated length. In the case of
data element sequences or alternates, this function calls itself recursively.

The maxLevel parameter specifies the maximum number of times this function calls itself. Limiting the recursion level prevents an infinite loop if the data is bad. maxLevel must be large enough to handle the complete data element. For example, to verify a simple data element such as an unsigned integer, maxLevel must be at least 1. To verify a data element sequence of UUIDs, maxLevel must be at least 2.

See Also BtLibSdpParseRawDataElement, BtLibSdpGetRawDataElementType, BtLibSdpGetRawDataElementSize

Application-Defined Functions

This section describes the callback functions that handle socket events. These functions are supplied by the developer and can be named anything.

New BtLibSocketCallback

Purpose Signal the result of a Bluetooth socket event. When the event takes place, this callback function is called.

Declared In BtLibTypes.h

Prototype void (*BtLibSocketProcPtr)(BtLibSocketEventType *sEvent, UInt32 refCon)

Parameters -> sEvent BtLibSocketEventType structure containing the event parameters.
-> refCon  General purpose integer which you can use to hold application-specific information. When you call BtLibSocketCreate to create the socket, you can specify a value to pass to this parameter.

**Result**  Returns nothing.

**Comments**  The event and status of the event are in the sEvent structure. See Socket Callback Events for more information.

You must specify this callback function when you create a socket. You do this by calling BtLibSocketCreate.
Cryptography Provider Manager

The Cryptography Provider Manager (CPM) is a shared library that acts as a framework for cryptographic services. These services are divided into two levels:

- At the application level, the CPM provides an API that any application can use to perform cryptographic operations.
- The operations themselves are not part of the CPM; instead, they’re provided by an algorithm provider (or AP). To export its functionality to the CPM, an algorithm provider implements a set of callback functions. Third party developers can create and distribute their own APs. The Palm OS provides a default algorithm provider that’s used in the absence of alternatives.

The CPM provides the glue between an application and an AP. In general, this glue is invisible: You don’t need to know anything about the APs that are available to use their cryptographic services. This makes it possible for an application to use common cryptographic operations without having to be overly concerned with the exact implementation of these operations. It also relieves the algorithm providers from the duty of providing their own invocation API.

**NOTE:** In Palm OS 5, only the application level API is offered; the ability to create and install your own algorithm provider will be supported in a later release.

The Default Provider

In Palm OS 5, there is only one algorithm provider. Implemented by RSA, it provides three cryptographic services:
• Data encryption and decryption using an RC4 symmetric key stream cipher
• Message digest creation (hashing) using the SHA-1 algorithm.
• Message verification through a combination of the two other operations.

All of the Palm OS 5 cryptography operations are provided by RSA.

One of the features of the CPM library is that the developer doesn’t need to know anything about cryptography. The functions are designed such that the default settings will yield satisfying results.

Fundamental CPM Functions

The fundamental cryptography functions are:

• CPMLibEncrypt performs a data encryption operation.
• CPMLibDecrypt performs a data decryption operation.
• CPMLibHash creates a message digest.
• CPMLibVerify verifies a message that has a signature and certificate.

These functions operate on blocks of data. For example, CPMLibEncrypt takes a block of data (and a key), encrypts it, and hands back the encrypted data all in one go.

The CPM also provides sets of functions that let you perform these operations sequentially, allowing you to (for example) initialize an encryption “stream,” iteratively feed data to the encryption algorithm, and then “finalize” the stream and get the encrypted data back. These functions are (for encryption) CPMLibEncryptInit, CPMLibEncryptUpdate, and CPMLibEncryptFinal.

Using the Crypto-Info Structures

The cryptography operations that are currently supported by the CPM library rely on four “crypto-info” data structures:

• The APKeyInfoStruct describes a symmetric key,
• **APCipherInfoStruct** describes an encryption/decryption operation.
• **APHashInfoStruct** describes message digest creation.
• **APVerifyInfoStruct** describes a message verification operation.

When you perform a cryptographic operation, you’ll be asked for one or more of these structures. In most cases you can pass in an “empty” structure and the function will populate it for you. However, it’s important that you zero the structure first.

For example, the **CPMLibHash** function takes a **APHashInfoStruct** as an argument. To use the default hashing operations, simply allocate the structure, zero it, and pass it in:

```c
APHashInfoType hashinfo;
MemSet ((void *)&hashinfo, sizeof (APHashInfoType), 0);
CPMLibHash( ..., &hashinfo, ...);
```

When you’re done with a crypto-info structure, you must “release” it:

```c
CPMLibReleaseHashInfo( ..., &hashinfo );
```

---

**Using the Export Functions**

The cryptography structures (key, hash, cipher, verification) can be “exported,” or encoded into a form that can be cached. These operations are provided by the **CPMLibExportObjectInfo** functions. Purposeful details of exporting (and importing, its functional complement) are given in the individual function descriptions, below. This little section touches on a wrinkle of usage.

When you use an export function, you’re asked to supply a buffer that can accommodate the encoded data. This means that you have to know how big the encoded data will be; the export function itself tells you the size through its final (reference) argument. Thus, you have to call the export function twice: Once to get the buffer size, and then again to actually get the encoded data. This is demonstrated below:
Cryptography Provider Manager

CPM and AP Constants

```c
UInt32 length=0;
UInt8 *data=NULL;
Err error;

/* For demo purposes, we’re only interested in the last two
 * arguments (the buffer and the buffer length). First, we
 * set the buffer to NULL, and retrieve the length.
 */
error = CPMLibExport(... NULL, &length);

/* At this pass, we expect the function to tell us that
 * the buffer is too small. Any other return is treated
 * as an error. Note that the <length> argument is reset to
 * to the required allocation length despite the error
 * return.
 */
if (error != cpmErrBufTooSmall)
    return ...; // or whatever

/* Allocate the buffer (for brevity, the example omits
 * the error check).
 */
data = MemPtrNew(length);

/* Call the export function again to retrieve the
 * encoded data.
 */
error = CPMLibExport(... ( ... data, &length);

/* This time we want a ‘clean’ return. */
if (error != errNone) {
    // handle the error
}
```

**CPM and AP Constants**

▶️ **New AP Capability Constants**

The AP capability values are bitfield constants that represent the capabilities that an AP supports. The capability constants are OR’d into the `flags` field of the `APProviderInfoStruct` structure.
Declared in `CPMLibCommon.h`, the constants are:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APF_MP</td>
<td>Streaming operations (initialize/update/finalize) are supported (“MP” stands for “multiple part”). See “Fundamental CPM Functions” for details. Note that all APs are expected to support block operations.</td>
</tr>
<tr>
<td>APF_HW</td>
<td>The AP’s algorithms are implemented in hardware (such as a SmartCard).</td>
</tr>
<tr>
<td>APF_KEYGEN</td>
<td>Symmetric key operations (including key info import and export) are supported.</td>
</tr>
<tr>
<td>APF_KEYPAIRGEN</td>
<td>Asymmetric key operations (including key info import and export) are supported.</td>
</tr>
<tr>
<td>APF_KEYDERIVE</td>
<td>Key derivation is supported.</td>
</tr>
<tr>
<td>APF_HASH</td>
<td>Hashing operations (including hash info import and export) are supported.</td>
</tr>
<tr>
<td>APF_CIPHER</td>
<td>Message encryption and decryption operations (including cipher info import and export) are supported.</td>
</tr>
<tr>
<td>APF_SIGN</td>
<td>Message signing operations (including sign info import and export) are supported.</td>
</tr>
<tr>
<td>APF_VERIFY</td>
<td>Message verification operations (including verify info import and export) are supported.</td>
</tr>
</tbody>
</table>
New Block Encryption Mode Constants

The constants listed below represent the various block encryption modes that may be supported by an AP. You can request a particular mode by setting the `APCipherInfoStruct.mode` field before passing the structure to the `CPMLibEncrypt` or `CPMLibDecrypt` function. If you don’t specify a mode, the AP will choose one for you, and reset the `mode` field to the chosen mode.

The `APModeEnum` data type is used to type the encryption mode constants.

```c
typedef UInt32 APModeEnum;
```

Note that encryption modes apply to block operations only. Specifying a mode for a stream encryption (through `CPMLibEncryptInit` et al.) has no effect.

The constants (and the type) are declared in `CPMLibConstants.h`. The constants are:

- `apModeTypeUnspecified = 0` The mode isn’t specified; the AP uses its default mode.
- `apModeTypeNone` The encryption mode doesn’t apply.
- `apModeTypeECB` Electronic codebook mode.
- `apModeTypeCBC` Cipher block chaining mode.
- `apModeTypeCBC_CTS` Cipher block chaining with cipher text stealing.
- `apModeTypeCFB` Cipher feedback mode.
- `apModeTypeOFB` Output feedback mode.
- `apModeCounter` Counter mode
Cipher Algorithm Constants

These constants represent the different cipher algorithms that an AP may support. The algorithm type is encoded in the type field of the APKeyInfoStruct structure. You can request a specific algorithm by setting the value of the type field before passing the structure to CPMLibGenerateKey. If you don’t care, leave the field zero’d (apAlgorithmTypeUnspecified); the function will set the field to tell you which algorithm was used.

In the APKeyInfoStruct structure, the constants are typed as APAlgorithmEnum:

```c
typedef UInt32 APAlgorithmEnum
```

The constants (and the type) are defined in CPMLibCommon.h. There we see the usual “unspecified” constant:

```c
apAlgorithmTypeUnspecified = 0    // No algorithm specified; the AP will use its default.
```

The rest of the constants are divided into groups, below, and listed with very little additional explanation: The constants’ names are reasonably self-documenting.

### Block Cipher Algorithms

- apSymmetricTypeDES
- apSymmetricTypeRC2
- apSymmetricTypeRC4
- apSymmetricTypeRC5
- apSymmetricTypeRC6,
- apSymmetricTypeDESX_XDX3 (“Strong” DES)
- apSymmetricType3DES_EDE2
- apSymmetricType3DES_EDE3
- apSymmetricTypeIDEA
- apSymmetricTypeDiamond2
- apSymmetricTypeBlowfish
- apSymmetricTypeTEA (Tiny Encryption Algorithm),
- apSymmetricTypeSAFER (Safe and Fast Encryption Routine),
Cryptography Provider Manager
CPM and AP Constants

apSymmetricType3WAY
apSymmetricTypeGOST (USSR Government Standard),
apSymmetricTypeSHARK
apSymmetricTypeCAST128, apSymmetricTypeSquare,
apSymmetricTypeSkipjack

Stream Ciphers

apSymmetricTypePanama, apSymmetricTypeARC4,
apSymmetricTypeSEAL, apSymmetricTypeWAKE,
apSymmetricTypeSapphire, apSymmetricTypeBBS

AES Block Ciphers

apSymmetricTypeRijndael, apSymmetricTypeCAST256,
apSymmetricTypeTwofish, apSymmetricTypeMARS,
apSymmetricTypeSerpent

Asymmetric Key Ciphers

apAsymmetricTypeRSA, apAsymmetricTypeDSA,
apAsymmetricTypeElgamal,
apAsymmetricTypeNR (Nyberg-Rueppel),
apAsymmetricTypeBlumGoldwasser,
apAsymmetricTypeRabin,
apAsymmetricTypeRW (Rabin-Williams),
apAsymmetricTypeLUC, apAsymmetricTypeLUCELG,
apAsymmetricTypeECDSA, apAsymmetricTypeECNR,
apAsymmetricTypeECIES, apAsymmetricTypeECDHC,
apAsymmetricTypeECMQVC

Key Agreement Ciphers

apKeyAgreementTypeDH (Diffie-Hellman),
apKeyAgreementTypeDH2 (Unified Diffie-Hellman),
apKeyAgreementTypeMQV (Menezes-Qu-Vanstone),
apKeyAgreementTypeLUCDIF, apKeyAgreementTypeXTRDH
New Export Encoding Constants

Constants that represent different data encoding schemes that are used to convert the crypto-info structs (APKeyInfoStruct, APHashInfoStruct, et al.) into a form that can be cached.

The encoding formats are used by the import and export functions, CPMLibImportKeyInfo, CPMLibExportKeyInfo, CPMLibImportHashInfo, CPMLibExportHashInfo, CPMLibImportVerifyInfo, CPMLibExportVerifyInfo, CPMLibImportCipherInfo, and CPMLibExportCipherInfo.

Declared in CPMLibCommon.h, the constants are:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMPORT_EXPORT_TYPE_RAW</td>
<td>The default encoding, as defined by the AP.</td>
</tr>
<tr>
<td>IMPORT_EXPORT_TYPE_DER</td>
<td>ASN.1 DER encoding.</td>
</tr>
<tr>
<td>IMPORT_EXPORT_TYPE_XML</td>
<td>Standardized XML encoding.</td>
</tr>
</tbody>
</table>

New Hashing Algorithm Constants

The hashing constants represent various hashing algorithms. If you want to tell the AP to use a specific algorithm in a hashing operation, you would pass one of these constants as an argument to CPMLibHash or CPMLibHashInit. If you want the default, use apHashTypeUnspecified, The algorithm that’s actually used is returned through a APHashInfoStruct structure.
Cryptography Provider Manager
CPM and AP Constants

Declared in CPMLibCommon.h, the constants are:

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>apHashTypeUnspecified</code></td>
<td>Unspecified hash algorithm; when generating a message digest, the AP decides which algorithm to use.</td>
</tr>
<tr>
<td><code>apHashTypeNone</code></td>
<td>Don’t hash. Use this constant if you want the AP to suppress hashing in an operation, such as verification, that normally performs it.</td>
</tr>
<tr>
<td><code>apHashTypeMD2</code></td>
<td>Rivest Message Digest 2 (MD2)</td>
</tr>
<tr>
<td><code>apHashTypeMD5</code></td>
<td>Rivest Message Digest 5 (MD5)</td>
</tr>
<tr>
<td><code>apHashTypeSHA1</code></td>
<td>Secure Hash Algorithm-160 (SHA-1)</td>
</tr>
<tr>
<td><code>apHashTypeSHA256</code></td>
<td>SHA 256-bit algorithm</td>
</tr>
<tr>
<td><code>apHashTypeSHA384</code></td>
<td>SHA 384-bit algorithm</td>
</tr>
<tr>
<td><code>apHashTypeSHA512</code></td>
<td>SHA 512-bit algorithm</td>
</tr>
<tr>
<td><code>apHashTypeHAVAL</code></td>
<td>HAVAL one-way algorithm</td>
</tr>
<tr>
<td><code>apHashTypeRIPEMD160</code></td>
<td>RIPEMD 160-bit algorithm</td>
</tr>
<tr>
<td><code>apHashTypeTiger</code></td>
<td>Tiger algorithm</td>
</tr>
<tr>
<td><code>apHashTypePanama</code></td>
<td>PANAMA algorithm</td>
</tr>
</tbody>
</table>

### Key Class Constants

A key’s “class” specifies whether the key is symmetric or asymmetric and, if the latter, whether it’s public or private. The key class constants represent these qualities. Key class is encoded in the `keyclass` field of the `APKeyInfoStruct` structure.

In the struct, the constants are typed as `APKeyClassEnum`: 
typedef UInt32 APKeyClassEnum;

The constants (and the type) are defined in CPMLibCommon.h. The constants are:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apKeyClassUnspecified = 0</td>
<td>The key’s class is unspecified.</td>
</tr>
<tr>
<td>apKeyClassSymmetric</td>
<td>This is a symmetric key.</td>
</tr>
<tr>
<td>apKeyClassPublic</td>
<td>This is the public part of an asymmetric key.</td>
</tr>
<tr>
<td>apKeyClassPrivate</td>
<td>This is the private part of an asymmetric key.</td>
</tr>
</tbody>
</table>

**New**

**Key Usage Constants**

**NOTE:** The key usage constants are currently unused.

The key usage constants describe the different ways that an encryption key can be used. How a key is used is encoded in the usage field of the key’s APKeyInfoStruct structure.

Note that the key usage values are mutually exclusive; you can’t OR a set of key usage constants to create a “selectively talented” key. The apKeyUsageAll constant is the only “multi-purpose” value currently provided.

The APKeyUsageEnum data type is used to type the key usage constants:

typedef UInt32 APKeyUsageEnum;

The constants (and the type) are defined in CPMLibConstants.h. The constants are

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apKeyUsageUnspecified = 0</td>
<td>The key’s usage is unspecified.</td>
</tr>
<tr>
<td>apKeyUsageAll</td>
<td>The key can be used in any operation.</td>
</tr>
</tbody>
</table>
Cryptography Provider Manager

CPM and AP Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apKeyUsageCertificateSigning</td>
<td>The key is intended for certificate signing.</td>
</tr>
<tr>
<td>apKeyUsageSigning</td>
<td>The key is intended for message signing operations.</td>
</tr>
<tr>
<td>apKeyUsageEncryption</td>
<td>The key is intended for key or data encryption operations.</td>
</tr>
<tr>
<td>apKeyUsageKeyEncrypting</td>
<td>The key is intended for key encryption operations.</td>
</tr>
<tr>
<td>apKeyUsageDataEncrypting</td>
<td>The key is intended for data encryption operations.</td>
</tr>
<tr>
<td>apKeyUsageMessageIntegrity</td>
<td>The key is intended for message verification operations.</td>
</tr>
</tbody>
</table>

New

Plaintext Padding Constants

The plaintext padding constants describe the different ways that plaintext is padded before it’s encrypted. The padding type is encoded in the padding field of an `APCipherInfoStruct` structure.

The `APPaddingEnum` data type is used to type the padding constants:
typedef UInt32 APPaddingEnum;

The constants (and the type) are declared in CPMLibCommon.h. The constants are:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apPaddingTypeUnspecified = 0</td>
<td>The padding is unspecified; the AP will use the default</td>
</tr>
<tr>
<td>apPaddingTypeNone</td>
<td>Specifically request that padding not be applied.</td>
</tr>
<tr>
<td>apPaddingTypePKCS1Type1</td>
<td>Public Key Cryptography Standard 1, type 1</td>
</tr>
<tr>
<td>apPaddingTypePKCS1Type2</td>
<td>Public Key Cryptography Standard 1, type 2</td>
</tr>
<tr>
<td>apPaddingTypePKCS5</td>
<td>Public Key Cryptography Standard 5</td>
</tr>
<tr>
<td>apPaddingTypeOAEP</td>
<td>Optimal Asymmetric Encryption Padding</td>
</tr>
<tr>
<td>apPaddingTypeSSLv23</td>
<td>Secure Sockets Layer version 23</td>
</tr>
</tbody>
</table>

**New**

**APCipherInfoStruct**

The APCipherInfoStruct encapsulates information about an instance of a data encryption or decryption operation. The structure is populated and returned by the data encryption and decryption functions (CPMLibEncrypt, CPMLibDecrypt, CPMLibEncryptInit, and CPMLibDecryptInit). You can set some of the fields’ values yourself (before calling an encryption/decryption function) to fine-tune the impending operation.
You’re responsible for allocating and freeing the APCipherInfoStructs that you need—the CPM never allocates them for you. The APCipherInfoStructs that you create (and actually use) must be released through CPMLibReleaseCipherInfo before they’re freed.

For more information (including examples) on how to use a crypto-info structure, see “Using the Crypto-Info Structures.”

Declared in CPMLibCommon.h, the structure looks like this:

```c
struct APCipherInfoStruct {
  APProviderContextType  providerContext;
  APAlgorithmEnum  type;
  APPaddingEnum  padding;
  UInt8  *iv;
  UInt32  ivLength;
  void  *algorithmParams;
};
```

The fields are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>providerContext</td>
<td>Information about the AP that performed (or is requested to perform) the operation. See APProviderContextStruct.</td>
</tr>
<tr>
<td>type</td>
<td>Constant that represents the cipher algorithm that was used or that’s requested. See Cipher Algorithm Constants</td>
</tr>
<tr>
<td>padding</td>
<td>Constant that represents the plaintext padding scheme used by the algorithm. See Plaintext Padding Constants</td>
</tr>
<tr>
<td>iv</td>
<td>Initialization vector.</td>
</tr>
<tr>
<td>ivLength</td>
<td>The length of the initialization vector, in bytes.</td>
</tr>
<tr>
<td>algorithmParams</td>
<td>Additional data that’s fed to the algorithm, as specified by the AP.</td>
</tr>
</tbody>
</table>
New

APHashInfoStruct

The APHashInfoStruct contains information about an instance of a message digest operation. You allocate the structure yourself, zero its contents, set the fields that you’re interested in (if any), and then pass it as an argument to the message digest functions (CPMLibHash, CPMLibHashInit, CPMLibExportHashInfo, and so on). When you’re finished with the struct, you pass it to CPMLibReleaseHashInfo.

For more information (including examples) on how to use the structure, see “Using the Crypto-Info Structures.”

Declared in CPMLibCommon.h, the structure looks like this:

```c
struct APHashInfoStruct {
    APProviderContextType  providerContext;
    APHashEnum  type;
    UInt32  length;
};
```

The fields are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>providerContext</td>
<td>Information about the AP that performed (or is requested to perform) the operation. See APProviderContextStruct.</td>
</tr>
<tr>
<td>type</td>
<td>The hashing algorithm that was used to create (or is requested to create) the message digest.</td>
</tr>
<tr>
<td>length</td>
<td>The length of the digest.</td>
</tr>
</tbody>
</table>

The APHashInfoStruct is one of the crypto-info structures; it’s used as an input/output argument to the message digest functions (CPMLibHash, CPMLibHashInit, CPMLibExportHashInfo, and so on). You allocate the structure yourself; before passing the struct to a function, you must zero its contents. When you’re finished with the struct, you pass it to CPMLibReleaseHashInfo. For more
information on how to use a crypto-info structure, see “Using the Crypto-Info Structures.”

\section*{\textbf{APKeyInfoStruct}}

The \texttt{APKeyInfoStruct} holds information about an encryption key.

\begin{verbatim}
struct APKeyInfoStruct {
    APProviderContextType  providerContext;
    APAlgorithmEnum       type;
    APKeyUsageEnum        usage;
    APKeyClassEnum        keyclass;
    UInt32 length;
    UInt32 actualLength;
    UInt16 exportable;
    UInt16 ephemeral;
};
\end{verbatim}

- **providerID**: Algorithm provider ID number.
- **type**: A code that identifies the type of algorithm, one of the Cipher Algorithm Constants values.
- **usage**: A code that identifies how the algorithm is used, one of the Key Usage Constants values.
- **keyLength**: Length of the key data, in bytes and padded to the next largest word. (Default is 8.)
- **keyActualLength**: The actual, unpadded length of the key data. (Default is 7.)
- **exportable**: Can this key be used in a CPMExportKey() call?. 1 if it can, 0 if it can’t.
- **ephemeral**: Is this a “one-shot” key? 1 if it is, 0 if it’s permanent.
**New**

**APProviderContextStruct**

The `APProviderContextStruct` contains information about an instance of a cryptographic operation. The structure is contained (as an `APProviderContextType` value) in the four crypto-info structs, `APKeyInfoStruct`, `APHashInfoStruct`, `APCipherInfoStruct`, and `APVerifyInfoStruct`.

Declared in `CPMLibCommon.h`, the structure looks like this:

```c
struct APProviderContextStruct {
    UInt32  providerID;
    void  *localContext;
};
```

The fields are:

- **providerID**: Integer that uniquely identifies the provider that’s being used in the operation.
- **localContext**: Provider-specific information about the operation.

**New**

**APProviderInfoStruct**

The `APProviderInfoStruct` contains information about a specific algorithm provider. The structure is returned (as an `APProviderInfoType`) by `CPMLibGetProviderInfo`.

Declared in `CPMLibCommon.h`, the structure looks like this:

```c
struct APProviderInfoStruct {
    char  name[32];
    char  other[64];
    UInt32  flags;
    UInt8  numAlgorithms;
    Boolean  bHardware;
};
```
The fields are

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The human-readable name of the provider.</td>
</tr>
<tr>
<td>other</td>
<td>Additional textual information.</td>
</tr>
<tr>
<td>flags</td>
<td>A bitfield that publishes the functionality that this provider supports.</td>
</tr>
<tr>
<td></td>
<td>See AP Capability Constants for a list of values that this field can combine.</td>
</tr>
<tr>
<td>numAlgorithms</td>
<td>A count of the algorithms this provider supplies.</td>
</tr>
<tr>
<td>bHardware</td>
<td>true if the provider’s algorithms are implemented in hardware; false if in software.</td>
</tr>
</tbody>
</table>

**New** APVerifyInfoStruct

The APVerifyInfoStruct is used by the verification functions (CPMLibVerify, et al.) to verify a message. It contains (primarily) the hash operation and cipher operation information that will be used during verification. It’s the caller’s responsibility to allocate and embed the structure’s APHashInfoStruct and APCipherInfoStruct fields before passing the APVerifyInfoStruct to a verification function.

Any APVerifyInfoStructs that you actually use must be released through CPMLibReleaseVerifyInfo before it’s freed. The embedded structures must also be released through CPMLibReleaseCipherInfo and CPMLibReleaseHashInfo.

For more information (including examples) on how to use a crypto-info structure, see “Using the Crypto-Info Structures.”

Declared in CPMLibCommon.h, the structure looks like this:

```c
struct APVerifyInfoStruct {
    APProviderContextType providerContext;
```
The fields are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>providerContext</td>
<td>Information about the AP that performed (or is requested to perform) the operation. See APProviderContextStruct.</td>
</tr>
<tr>
<td>hashInfoP</td>
<td>APHashInfoStruct that contains the certificate’s hash information.</td>
</tr>
<tr>
<td>cipherInfoP</td>
<td>APCipherInfoStruct that contains the certificate’s cipher information.</td>
</tr>
</tbody>
</table>

**New**

**CPMInfoStruct**

Structure that provides information about the CPM library. It’s used by the CPMLibGetInfo function.

```c
typedef struct CPMInfoStruct {
    UInt8  numInstances;
    UInt8   numProviders;
    Boolean  defaultProviderPresent;
} CPMInfoStruct;
```

The fields are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numInstances</td>
<td>The number of clients (applications) that are talking to this library.</td>
</tr>
<tr>
<td>numProviders</td>
<td>The number of algorithm providers that this library knows about.</td>
</tr>
<tr>
<td>defaultProviderPresent</td>
<td>Does the library contain a default provider? true if it does, otherwise false.</td>
</tr>
</tbody>
</table>
CPM Functions

New

CPMLibDecrypt

Purpose
Decrypts a block of encrypted data.

Declared In
CPMLib68kInterface.h, CPMLibARMInterface.h

Prototype
Err CPMLibDecrypt ( UInt16 libRef,
APKeyInfoType *keyInfo,
APCIPHERINFOType *cipherInfo, UInt8 *inBuffer,
UInt32 inBufferLength, UInt8 *outBuffer,
UInt32 *outBufferLength )

Parameters
-> libRef (68k only) CPM Library reference number.

-> keyInfo APKeyInfoStruct that contains the key that
was used to encrypt the data. You obtain the
structure by importing the APKeyInfoStruct
that was exported by the data encryptor.

<- cipherInfo
A pointer to an APCipherInfoStruct that
describes the parameters that will be used in
the decryption operation. You typically retrieve
the structure by importing the APCipherInfoStruct that was exported by
the data encryptor. If you’re using the
decryption defaults provided by the CPM
library, you can pass in a freshly allocated (and
zero’d) APCipherInfoStruct; in this case,
the structure will be populated by this function
to reflect the actual decryption parameters.

-> inBuffer A pointer to the (encrypted) data that you want
to decrypt.

-> inBufferLength Length, in bytes, of inBuffer.
<- outBuffer  A pointer to the buffer into which the
decrypted data will be copied. The buffer must
be allocated by the caller, and must be big
enough to accommodate all of the decrypted
data.

<-> outBufferLength

You pass in the (allocated) length of
outBuffer, in bytes. The function resets the
argument to the amount of data that was
actually copied into outBuffer. If the function
returns cpmErrBufTooSmall,
outBufferLength is set to the minimum
buffer size that’s needed to accommodate the
decrypted data..

Result  The function returns errNone upon success. For other error codes,
see CPM Error Codes.

Comments  This functions peforms block decryption. For stream decryption, see
CPMLibDecryptInit.

The keyInfo and cipherInfo must agree on the algorithm type,
as specified in their respective APAlgorithmEnum fields.

If outBuffer isn’t big enough, the function will fail and
outBufferLength will return the “correct” output buffer size (i.e. large
enough to accommodate the encrypted data). When this
happens, simply reallocate the output buffer and call
CPMLibDecrypt again.
Cryptography Provider Manager
CPM Functions

New

CPMLibDecryptFinal

Purpose Finalizes a stream decryption operation.

Declared In CPMLib68kInterface.h, CPMLibARMInterface.h

Prototype

Err CPMLibDecryptFinal ( UInt16 libRef, 
 APKeyInfoType *keyInfo,
 APCipherInfoType *cipherInfo, UInt8 *inBuffer,
 UInt32 inBufferLength, UInt8 *outBuffer,
 UInt32 *outBufferLength) 

Parameters

-> libRef CPM Library reference number (68k only).

-> keyInfo Key used to decrypt the data, as returned by 
 CPMLibGenerateKey or 
 CPMLibImportKeyInfo.

-> cipherInfo A pointer to the APCipherInfoStruct that 
 was returned by the stream-initializing 
 CPMLibDecryptInit call.

-> inBuffer A pointer to the data that you want to encrypt. 
 If you already supplied all the data through 
 previous CPMLibDecryptUpdate calls, pass 
 NULL.

-> inBufferLength The length of inBuffer, in bytes. If inBuffer 
 is NULL, pass 0.

<- outBuffer A pointer to the buffer where the decrypted 
 inBuffer data will be copied. The buffer must 
 be allocated by the caller, and must be big 
 enough to accommodate all of the decrypted 
 data.
<-> outBufferLength
You pass in the (allocated) length of outBuffer, in bytes. The function resets the argument to the amount of data that was actually copied into outBuffer. If the function returns cpmErrBufTooSmall, outBufferLength is set to the minimum buffer size that’s needed to accommodate the decrypted inBuffer data.

Result
The function returns errNone upon success. For other error codes, see CPM Error Codes.

Comments
This function (optionally) decrypts a final buffer of data and then closes the decryption stream that was initialized by CPMLibDecryptInit.

Note that outBuffer contains the decrypted inBuffer data only—it doesn’t contain all the data that was decrypted by this stream. It’s the caller’s responsibility to accumulate the data that was decrypted by previous, successive CPMLibDecryptUpdate calls.

New

CPMLibDecryptInit

Purpose
Initializes a stream decryption session.

Declared In
CPLib68kInterface.h, CPMLibARMInterface.h

Prototype
Err CPMLibDecryptInit ( UInt16 libRef, APKeyInfoType *keyInfo, APCipherInfoType *cipherInfo )

Parameters
- -> libRef CPM Library reference number (68k only).
- -> keyInfo APKeyInfoStruct that contains the key that was used to encrypt the data. You obtain the structure by importing the APKeyInfoStruct that was exported by the data encryptor.
<-> cipherInfo

A pointer to an APCipherInfoStruct that describes the parameters that will be used in the decryption operation. You typically retrieve the structure by importing the APCipherInfoStruct that was exported by the data encryptor. If you’re using the decryption defaults provided by the CPM library, you can pass in a freshly allocated (and zero’d) APCipherInfoStruct; in this case, the structure will be populated by this function to reflect the actual decryption parameters.

**Result**
The function returns errNone upon success. For other error codes, see CPM Error Codes.

**Comments**
This function initializes a stream decryption operation. To feed data to the operation, you call CPMLibDecryptUpdate followed by CPMLibDecryptFinal. The “update” function is optional; the “final” function is mandatory. For block encryption, see CPMLibDecrypt.

The keyInfo and cipherInfo must agree on the algorithm type, as specified in their respective APAlgorithmEnum fields.
**New**  

**CPMLibDecryptUpdate**  

Feeds data to a stream decryption operation.

**Declared In**  

CPLib68kInterface.h, CPMLibARMInterface.h

**Prototype**  

Err CPMLibDecryptUpdate ( UInt16 libRef,  
APKeyInfoType *keyInfo,  
APCipherInfoType *cipherInfo, UInt8 *inBuffer,  
UInt32 inBufferLength, UInt8 *outBuffer,  
UInt32 *outBufferLength)

**Parameters**  

see CPMLibEncryptFinal

**Result**  

The function returns errNone upon success. For other error codes, see CPM Error Codes.

**Comments**  

This function feeds data into the stream decryption session that was started by CPMLibDecryptInit. You can make any number of CPMLibDecryptUpdate calls while the stream is open. When you’ve finished feeding data into the stream, you call CPMLibDecryptFinal.

This function’s arguments, return values, and behavior are nearly identical to CPMLibDecryptFinal (which see for details). The only difference between them is that this function leaves the stream open, and CPMLibDecryptFinal closes it.
**New**

**CPMLibEncrypt**

**Purpose**
Encrypts a block of data.

**Declared In**
CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype**
```
Err CPMLibEncrypt ( UInt16 libRef,
                   APKeyInfoType *keyInfo,
                   APCipherInfoType *cipherInfo, UInt8 *inBuffer,
                   UInt32 inBufferLength, UInt8 *outBuffer,
                   UInt32 *outBufferLength)
```

**Parameters**
- `-> libRef`  
  CPM Library reference number (68k only).

- `-> keyInfo`  
  A pointer to an `APKeyInfoStruct` that represents the key that will be used to encrypt the data. The allocate the structure, zero it, and then populate it by calling `CPMLibGenerateKey` or `CPMLibImportKeyInfo`.

- `-> cipherInfo`  
  A pointer to an `APCipherInfoStruct` that you can use to set the parameters of the encryption operation. If the CPM can’t satisfy the requirements you specify in the structure, the operation will fail. If you want to use the default cipher settings, pass in a zero’d structure. When the function returns, the structure will be filled with information describing the operation.

- `-> inBuffer`  
  A pointer to the data that you want to encrypt.

- `-> inBufferLength`  
  The length of `inBuffer`, in bytes.

- `<- outBuffer`  
  A pointer to the buffer where the encrypted data will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate all of the encrypted data.
<-> outBufferLength

You pass in the (allocated) length of outBuffer, in bytes. The function resets the argument to the amount of data that was actually copied into outBuffer. If the function returns cpmErrBufTooSmall, outBufferLength is set to the minimum buffer size that’s needed to accommodate the encrypted data.

**Result**
The function returns errNone upon success. For other error codes, see CPM Error Codes.

**Comments**
This functions performs block encryption. For stream encryption, see CPMLibEncryptInit.

The keyInfo and cipherInfo must agree on the algorithm type, as specified in their respective APAlgorithmEnum fields.

If outBuffer isn’t big enough, the function will fail and outBufferLength will return the “correct” output buffer size (i.e. large enough to accommodate the encrypted data). When this happens, simply reallocate the output buffer and call CPMLibEncrypt again.

After you’ve encrypted the data, you must export the keyInfo and cipherInfo structures (see CPMLibExportKeyInfo and CPMLibExportCipherInfo) so they can be imported, later, by the data decryptor. Secure storage and transmission of the encryption key is the caller’s responsibility.

To decrypt encrypted data, you call CPMLibDecrypt or CPMLibDecryptInit.
New

CPMLibEncryptFinal

Purpose
Finalizes a stream encryption operation.

Declared In
CPMLib68kInterface.h, CPMLibARMInterface.h

Prototype
Err CPMLibEncryptFinal ( UInt16 libRef,
APKeyInfoType *keyInfo,
APCipherInfoType *cipherInfo, Uint8 *inBuffer,
Uint32 inBufferLength, Uint8 *outBuffer,
Uint32 *outBufferLength)

Parameters
-> libRef
CPM Library reference number (68k only).

-> keyInfo
Key used to encrypt the data, as returned by CPMLibGenerateKey or CPMLibImportKeyInfo.

-> cipherInfo
A pointer to the APCipherInfoStruct that was returned in the stream-initializing CPMLibEncryptInit call.

-> inBuffer
A pointer to the data that you want to encrypt. If you already supplied all the data through previous CPMLibEncryptUpdate calls, pass NULL.

-> inBufferLength
The length of inBuffer, in bytes. If inBuffer is NULL, pass 0.

<- outBuffer
A pointer to the buffer where the encrypted inBuffer data will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate the encrypted inBuffer data.
Cryptography Provider Manager

CPM Functions

<-> outBufferLength

You pass in the (allocated) length of outBuffer, in bytes. The function resets the argument to the amount of data that was actually copied into outBuffer. If the function returns cpmErrBufTooSmall, outBufferLength is set to the minimum buffer size that’s needed to accommodate the encrypted data.

Result

The function returns errNone upon success. For other error codes, see CPM Error Codes.

Comments

This function (optionally) encrypts a final buffer of data and then closes the encryption stream that was initialized by CPMLibEncryptInit.

After you’ve encrypted the data, you should export the keyInfo and cipherInfo structures (see CPMLibExportKeyInfo and CPMLibExportCipherInfo) so they can be imported, later, by the data decryptor. Secure storage and transmission of the encryption key is the caller’s responsibility.

Note that outBuffer contains the encrypted inBuffer data only—it doesn’t contain all the data that was encrypted by this stream. It’s the callers responsibility to accumulate the data that was encrypted by previous, successive CPMLibEncryptUpdate calls.

To decrypt encrypted data, you call CPMLibDecrypt or CPMLibDecryptInit.
New  CPMLibEncryptInit

Purpose  Initializes a stream encryption session.

Declared In  CPMLib68kInterface.h, CPMLibARMInterface.h

Prototype  Err CPMLibEncryptInit ( UInt16 libRef,
APKeyInfoType *keyInfo,
APCipherInfoType *cipherInfo )

Parameters
- > libRef  CPM Library reference number (68k only).
- > keyInfo  Key used to encrypt the data, as returned by
CPMLibGenerateKey or
CPMLibImportKeyInfo.
- > cipherInfo  A pointer to an APCipherInfoStruct that
you can use to set the parameters of the
encryption operation. If the CPM can’t satisfy
the requirements you specify in the structure,
the operation will fail. If you want to use the
default cipher settings, pass in a zero’d
structure. When the function returns, the
structure will be filled with information
describing the operation. The structure is used
as a cookie in the subsequent
CPMLibEncryptUpdate and/or
CPMLibEncryptFinal functions.

Result  The function returns errNone upon success. For other error codes,
see CPM Error Codes.

Comments  This function initializes a stream encryption operation. To feed data
to the operation, you call CPMLibEncryptUpdate followed by
CPMLibEncryptFinal. The “update” function is optional; the
“final” function is mandatory. For block encryption, see
CPMLibEncrypt.
The `keyInfo` and `cipherInfo` must agree on the algorithm type, as specified in their respective `APAlgorithmEnum` fields.

To decrypt encrypted data, you call `CPMLibDecrypt` or `CPMLibDecryptInit`.

---

**New**

**CPMLibEncryptUpdate**

**Purpose**

Feeds data to a stream encryption operation.

**Declared In**

`CPMLib68kInterface.h, CPMLibARMInterface.h`

**Prototype**

```
Err CPMLibEncryptUpdate ( UInt16 libRef, 
APKeyInfoType *keyInfo, 
APCipherInfoType *cipherInfo, UInt8 *inBuffer, 
UInt32 inBufferLength, UInt8 *outBuffer, 
UInt32 *outBufferLength)
```

**Parameters**

see `CPMLibEncryptFinal`

**Result**

The function returns `errNone` upon success. For other error codes, see `CPM Error Codes`.

**Comments**

This function feeds data into the stream encryption session that was started by `CPMLibEncryptInit`. You can make any number of `CPMLibEncryptUpdate` calls while the encryption stream is open. When you’ve finished feeding data into the stream, you call `CPMLibEncryptFinal`.

This function’s arguments, return values, and behavior are nearly identical to `CPMLibEncryptFinal` (which see for details). The only difference between them is that this function leaves the stream open, and `CPMLibEncryptFinal` closes it.
New CPMLibExportCipherInfo

Purpose  Encodes a cipher into a form that can be cached. To reconstitute an exported cipher, pass it to CPMLibImportCipherInfo.

Declared In  CPMLib68KInterface.h, CPMLibArmInterface.h

Prototype  Err CPMLibExportCipherInfo ( UInt16 libRef, APCipherInfoType *cipherInfo, UInt8 encoding, UInt8 *exportBuffer, UInt32 exportBufferLength )

Parameters
- libRef  CPM Library reference number (68k only).
- cipherInfo  Structure that represents the cipher that you want to export, as created and returned by CPMLibGenerateKey, or as imported through CPMLibImportKeyInfo.
- encoding  Constant that specifies the type of encoding. One of IMPORT_EXPORT_RAW, IMPORT_EXPORT_DER, or IMPORT_EXPORT_XML. See Export Encoding Constants for details about these formats.
- exportBuffer  Buffer into which the function copies the encoded data. The buffer must be allocated by the caller. Point this argument to NULL if you’re using the function to retrieve the size of the encoded data.
- exportBufferLength  You pass in the size of exportBuffer in bytes. The function returns (through this argument) the size that’s required to accommodate the encoded data.

Result  The function returns errNone upon success. For other error codes, see CPM Error Codes.
Comments
You call this function twice: Once to get the size of the required exportBuffer, and then again (after allocating the buffer) to get the encoded buffer. See “Using the Export Functions” for more information and a free sample.

New
CPMLibExportHashInfo

Purpose
Encodes an APHashInfoStruct into a form that can be cached. To reconstitute an exported hash info, pass it to CPMLibImportHashInfo.

Declared In
CPMLib68KInterface.h, CPMLibArmInterface.h

Prototype
Err CPMLibExportKeyInfo ( UInt16 libRef,
APHashInfoType *hashInfo, UInt8 encoding,
UInt8 *exportBuffer, UInt32 exportBufferLength )

Parameters
- `libRef` CPM Library reference number (68k only).
- `hashInfo` Structure that you want to export.
- `encoding` Constant that specifies the type of encoding. One of IMPORT_EXPORT_RAW, IMPORT_EXPORT_DER, or IMPORT_EXPORT_XML. See Export Encoding Constants for details about these formats.
- `exportBuffer` Buffer into which the function copies the encoded data. The buffer must be allocated by the caller. Point this argument to NULL if you’re using the function to retrieve the size of the encoded data (see below.
<-> exportBufferLength
You pass in the size of exportBuffer in bytes; the function returns (through this argument) the size that’s required to accommodate the encoded data.

**Result**
The function returns errNone upon success. For other error codes, see [CPM Error Codes](#).

**Comments**
You call this function twice: Once to get the size of the export buffer, and then again (after allocating the buffer) to retrieve the encoded data. See “Using the Export Functions” for more information and a free sample.

---

**New**

**CPMLibExportKeyInfo**

**Purpose**
Encodes an APKeyInfoStruct into a form that can be cached. To reconstitute an exported key, pass it to [CPMLibImportKeyInfo](#).

**Declared In**
CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype**
Err CPMLibExportKeyInfo ( Uint16 libRef, APKeyInfoType *keyInfo, Uint8 encoding, Uint8 *exportBuffer, Uint32 exportBufferLength )

**Parameters**
- `-> libRef` CPM Library reference number (68k only).
- `-> keyInfo` Structure that you want to export.
- `-> encoding` Constant that specifies the type of encoding. One of IMPORT_EXPORT_RAW, IMPORT_EXPORT_DER, or IMPORT_EXPORT_XML, See [Export Encoding Constants](#) for details about these formats.
<-> exportBuffer
Buffer into which the function copies the encoded data. The buffer must be allocated by the caller. Point this argument to NULL if you’re using the function to retrieve the size of the encoded data (see below)

<-> exportBufferLength
You pass in the size of exportBuffer in bytes; the function returns (through this argument) the size that’s required to accommodate the encoded data.

Result
The function returns errNone upon success. For other error codes, see CPM Error Codes.

Comments
You call this function twice: Once to get the size of the export buffer, and then again (after allocating the buffer) to retrieve the encoded data. See “Using the Export Functions” for more information and a free sample.

New
CPMLibExportVerifyInfo

Purpose
Encodes a APVerifyInfoStruct into a form that can be cached. To reconstitute an exported key, pass it to CPMLibImportVerifyInfo.

Declared In
CPMLib68KInterface.h, CPMLibArmInterface.h

Prototype
Err CPMLibExportKeyInfo ( UInt16 libRef, APVerifyInfoType *verifyInfo, UInt8 encoding, Uint8 *exportBuffer, Uint32 exportBufferLength )

Parameters
- > libRef CPM Library reference number (68k only).
- > keyInfo Structure that you want to export.
Cryptography Provider Manager

CPM Functions

-> encoding  Constant that specifies the type of encoding. One of IMPORT_EXPORT_RAW, IMPORT_EXPORT_DER, or IMPORT_EXPORT_XML. See Export Encoding Constants for details about these formats.

<-> exportBuffer
Buffer into which the function copies the encoded data. The buffer must be allocated by the caller. Point this argument to NULL if you’re using the function to retrieve the size of the encoded data (see below)

<-> exportBufferLength
You pass in the size of exportBuffer in bytes; the function returns (through this argument) the size that’s required to accommodate the encoded data.

Result  The function returns errNone upon success. For other error codes, see CPM Error Codes.

Comments  You call this function twice: Once to get the size of the export buffer, and then again (after allocating the buffer) to retrieve the encoded data. See “Using the Export Functions” for more information and a free sample.
Cryptography Provider Manager

CPM Functions

New

CPMLibGenerateKey

Purpose
Generates a new symmetric key.

Declared In
CPMLib68KInterface.h, CPMLibArmInterface.h

Prototype
Err CPMLibGenerateKey ( UInt16 libRef,
UInt8 *seedData, UInt32 seedLength,
APKeyInfoType *keyInfo )

Parameters
-> libRef
CPM Library reference number (68k only).

-> seedData
Optional data that’s used to seed the key generator. Because PalmOS 5 doesn’t currently support key “derivation” (identical key generation based on identical seeds) the seed data is, in essence, a no-op, and can be a pointer to 0. However, you may want to supply (and cache) unique seed data today in anticipation of tomorrow’s derivation functionality.

-> seedLength
Length of seedData, in bytes. (Pass 0 if *seedData is 0.)

<-> keyInfo
A pointer to a APKeyInfoStruct that will contain the generated key. The keyInfo structure is allocated and owned by the caller. You can specify the desired provider, key-generation scheme, and so on, by setting the APKeyInfoType fields. Or, to retrieve a default key, zero the structure before you pass it in.

Result
The function returns errNone upon success. For other error codes, see CPM Error Codes.

Comments
The APKeyInfoStruct that’s populated by this function can be used in subsequent encryption, decryption, and verification operations.
When you’re finished using the `APKeyInfoStruct`, you must release it through `CPMLibReleaseKeyInfo`.

---

**New** CPMLibGetInfo

**Purpose** Retrieves information about the CPM library.

**Declared In** `CPMLib68KInterface.h`, `CPMLibArmInterface.h`

**Prototype**

```
Err CPMLibLibGetInfo ( UInt16 libRef,
CPMInfoType *info)
```

**Parameters**

- `-> libRef` CPM Library reference number (68k only).
- `-> info` Pointer to a `CPMInfoStruct` that returns the information about the library. See `CPMInfoStruct` for a description.

**Result** The function returns `errNone` upon success. For other error codes, see `CPM Error Codes`.

---

**New** CPMLibGetProviderInfo

**Purpose** Retrieves information about a specific provider.

**Declared In** `CPMLib68KInterface.h`, `CPMLibArmInterface.h`

**Prototype**

```
Err CPMLibGetProviderInfo ( UInt16 libRef,
UInt32 providerID,
APProviderInfoType *providerInfo)
```

**Parameters**

- `-> libRef` CPM Library reference number (68k only).
- `-> providerID` ID number that identifies the provider.
<- providerInfo
Structure into which the function places provider information. The structure is allocated and freed by the caller. The function doesn’t clear inapplicable fields on the way out.

Result
The function returns errNone upon success. For other error codes, see CPM Error Codes.

New
CPMLibHash

Purpose
Hashes a block of data.

Declared In
CPMLib68KInterface.h, CPMLibArmInterface.h

Prototype
Err CPMLibHash ( UInt16 libRef, AHashEnum type, AHashInfoType *hashInfo, UInt8 *inBuffer, UInt32 inBufferLength, UInt8 *outBuffer, UInt32 *outBufferLength )

Parameters
-> libRef
CPM Library reference number (68k only).

-> type
A constant that represents the hashing algorithm that will be used to create the message digest. See Hashing Algorithm Constants for a list of constants. If you want the default algorithm, use apHashTypeUnspecified.

-> hashInfo
A pointer to an AHashInfoStruct that you can use to set the parameters of the hashing operation. If the CPM can’t satisfy the requirements you specify in the structure, the operation will fail. If you want to use the default settings, pass in a zero’d structure. When the function returns, the structure will be filled with information describing the operation.
Cryptography Provider Manager

CPM Functions

-> inBuffer A pointer to the data that you want to hash.
-> inBufferLength
   The length of inBuffer, in bytes. If inBuffer is NULL, pass 0.
<- outBuffer A pointer to the buffer where the hashed inBuffer data will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate all of the hashed data.
<- outBufferLength
   You pass in the (allocated) length of outBuffer, in bytes. The function resets the argument to the amount of data that was actually copied into outBuffer. If the function returns cpmErrBufTooSmall, outBufferLength is set to the minimum buffer size that’s needed to accommodate the hashed data.

Result
The function returns errNone upon success. For other error codes, see CPM Error Codes.

Comments
This function performs a block hash operation. For stream hashing, use CPMLibHashInit.

New

CPMLibHashFinal

Purpose
Finalizes a hash session and returns the hashed data.

Declared In
CPMLib.h

Prototype
Err CPMLibHashFinal ( UInt16 libRef, APHashInfoType *hashInfo, UInt8 *inBuffer, UInt32 inBufferLength, UInt8 *outBuffer, UInt32 *outBufferLength )

Parameters
-> libRef CPM Library reference number (68k only).
-> hashInfo
   A pointer to the APHashInfoStruct that was returned by CPMLibHashInit.

-> inBuffer
   A pointer to the data that you want to hash. If you already supplied all the data through
   previous CPMLibHashUpdate calls, pass NULL.

-> inBufferLength
   The length of inBuffer, in bytes. If inBuffer is NULL, pass 0.

<- outBuffer
   A pointer to the buffer where all the data that has been hashed by this stream will be copied.
   The buffer must be allocated by the caller, and must be big enough to accommodate all of the
   hashed data.

<-> outBufferLength
   You pass in the (allocated) length of outBuffer, in bytes. The function resets the argument
   to the amount of data that was actually copied into outBuffer. If the function returns
cpmErrBufTooSmall, outBufferLength is set to the minimum buffer size that’s needed to accommodate the
   hashed data.

**Result**

The function returns errNone upon success. For other error codes, see CPM Error Codes.

**Comments**

This function returns all the data that was hashed by the hash stream, and then closes the stream. It follows an initial call to
CPMLibHashInit and some number of calls to CPMLibHashUpdate.

---

**New**

CPMLibHashInit

**Purpose**

Initiates a streaming hash operation.
Cryptography Provider Manager  
CPM Functions

Declared In  CPMLib68KInterface.h, CPMLibArmInterface.h

Prototype  Err CPMLibHash ( UInt16 libRef,  
APHashInfoType *hashInfo )

Parameters  -> libRef  CPM Library reference number (68k only).
-> hashInfo  A pointer to an APHashInfoStruct that you can use to set the parameters of the hash operation. If the CPM can’t satisfy the requirements you specify in the structure, the operation will fail. If you want to use the default settings, pass in a zero’d structure. When the function returns, the structure will be filled with information describing the operation.

Result  The function returns errNone upon success. For other error codes, see CPM Error Codes.

Comments  This function initializes a streaming hash operation. To feed data to the stream, you call CPMLibHashUpdate followed by CPMLibHashFinal. The “update” function is optional; the “final” function is mandatory. For a block hash, see CPMLibHash.

\[\text{New}\]

cpmlibhashupdate

Purpose  Sends data to streaming hash operation.

Declared In  CPMLib68KInterface.h, CPMLibArmInterface.h

Prototype  Err CPMLibHashUpdate ( UInt16 libRef,  
APHashInfoType *hashInfo, UInt8 *inBuffer,  
UInt32 inBufferLength)

Parameters  -> libRef  CPM Library reference number (68k only).
-> hashInfo A pointer to the APhashInfoStruct that was returned by CPMLibHashInit.

-> inBuffer A pointer to the data that you want to hash.

-> inBufferLength The length of inBuffer, in bytes.

Result The function returns errNone upon success. For other error codes, see CPM Error Codes.

Comments This function feeds data into the streaming hash session that was started by CPMLibHashInit. You can make any number of CPMLibHashUpdate calls while the hash stream is open. When you’ve finished feeding data into the stream, you call CPMLibHashFinal.

Note that this function doesn’t return any hashed data. All the data that’s hashed by the stream is returned through the CPMLibHashFinal call.

New CPMLibImportCipherInfo

Purpose Imports a previously-exported cipher info structure so that it can be used in subsequent operations.

Declared In CPMLib68KInterface.h, CPMLibArmInterface.h

Prototype Err CPMLibImportCipherInfo ( UInt16 libRef, UInt8 encoding, UInt8 *importData, UInt32 importDataLength, APCipherInfoType *cipherInfo )

Parameters -> libRef CPM Library reference number (68k only).
Cryptography Provider Manager
CPM Functions

-> encoding  Constant that specifies the encoding type that was used to export the key. One of IMPORT_EXPORT_RAW, IMPORT_EXPORT_DER, or IMPORT_EXPORT_XML. See Export Encoding Constants for details about these formats.

-> importData  The encoded data.

-> importDataLength  The length of importData, in bytes.

<- cipherInfo  APCipherInfoStruct that returns the imported key. The structure must be allocated before it’s passed in.

Result  The function returns errNone upon success. For other error codes, see CPM Error Codes.

New  CPMLibImportHashInfo

Purpose  Imports a previously-exported hash info structure so that it can be used in subsequent operations.

Declared In  CPMLib68KInterface.h, CPMLibArmInterface.h

Prototype  Err CPMLibImportHashInfo ( UInt16 libRef, UInt8 encoding, UInt8 *importData, UInt32 importDataLength, AHashInfoType *keyInfo )

Parameters  -> libRef  CPM Library reference number (68k only).
-> encoding Constant that specifies the encoding type that was used to export the key. One of IMPORT_EXPORT_RAW, IMPORT_EXPORT_DER, or IMPORT_EXPORT_XML. See Export Encoding Constants for details about these formats.

-> importData The encoded data.

-> importDataLength The length of importData, in bytes.

<- hashInfo APHashInfoStruct that returns the imported hash info. The structure must be allocated before it’s passed in.

Result The function returns errNone upon success. For other error codes, see CPM Error Codes.

\[\]

New CPMLibImportKeyInfo

Purpose Imports a previously-exported key so that it (the key) can be used in subsequent operations.

Declared In CPMLib68KInterface.h, CPMLibArmInterface.h

Prototype Err CPMLibImportKeyInfo ( UInt16 libRef, UInt8 encoding, UInt8 *importData, UInt32 importDataLength, APKeyInfoType *keyInfo )

Parameters

-> libRef CPM Library reference number (68k only).

-> encoding Constant that specifies the encoding type that was used to export the key. One of IMPORT_EXPORT_RAW, IMPORT_EXPORT_DER, or IMPORT_EXPORT_XML. See Export Encoding Constants for details about these formats.
Cryptography Provider Manager
CPM Functions

New CPMLibImportVerifyInfo

Purpose Imports a previously-exported verify info structure so that it can be used in subsequent operations.

Declared In CPMLib68KInterface.h, CPMLibArmInterface.h

Prototype Err CPMLibImportVerifyInfo ( UInt16 libRef, UInt8 encoding, UInt8 *importData, UInt32 importDataLength, APVerifyInfoType *keyInfo )

Parameters
-> libRef CPM Library reference number (68k only).
-> encoding Constant that specifies the encoding type that was used to export the key. One of IMPORT_EXPORT_RAW, IMPORT_EXPORT_DER, or IMPORT_EXPORT_XML. See Export Encoding Constants for details about these formats.
-> importData The encoded data.
-> importDataLength The length of importData, in bytes.

Result The function returns errNone upon success. For other error codes, see CPM Error Codes.
<- verifyInfo APVerifyInfoStruct that returns the imported verify info. The structure must be allocated before it’s passed in.

**Result**
The function returns errNone upon success. For other error codes, see [CPM Error Codes](#).

---

### New CPMLibReleaseCipherInfo

**Purpose**
Releases a APCipherInfoStruct, allowing you to free it.

**Declared In**
CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype**
Err CPMLibReleaseCipherInfo ( UInt16 libRef, APCipherInfoType *cipherInfo )

**Parameters**
- **libRef**
  CPM Library reference number (68k only).
- **cipherInfo**
  A pointer to a APCipherInfoStruct that you want to release.

**Result**
The function returns errNone upon success. For other error codes, see [CPM Error Codes](#).

---

### New CPMLibReleaseHashInfo

**Purpose**
Releases a APKeyInfoStruct, allowing you to free it.

**Declared In**
CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype**
Err CPMLibReleaseHashInfo ( UInt16 libRef, APHashInfoType *hashInfo )

**Parameters**
- **libRef**
  CPM Library reference number (68k only).
New CPMLibReleaseKeyInfo

Purpose
Releases an APKeyInfoStruct, allowing you to free it.

Declared In
CPMLib68KInterface.h, CPMLibArmInterface.h

Prototype
Err CPMLibReleaseKeyInfo ( UInt16 libRef, APKeyInfoType *keyInfo )

Parameters
- `libRef` CPM Library reference number (68k only).
- `keyInfo` A pointer to the APKeyInfoStruct that you want to release.

Result
The function returns errNone upon success. For other error codes, see CPM Error Codes.

New CPMLibReleaseVerifyInfo

Purpose
Releases a APVerifyInfoStruct, allowing you to free it.

Declared In
CPMLib68KInterface.h, CPMLibArmInterface.h

Prototype
Err CPMLibReleaseVerifyInfo ( UInt16 libRef, APVerifyInfoType *verifyInfo )

Parameters
- `libRef` CPM Library reference number (68k only).

Result
The function returns errNone upon success. For other error codes, see CPM Error Codes.
CPMLibVerify

Purpose
Verifies a message.

Declared In
CPMLib68KInterface.h, CPMLibArmInterface.h

Prototype
Err CPMLibVerify ( UInt16 libRef,
APKeyInfoType *keyInfo,
APVerifyInfoType *verifyInfo, UInt8 *inBuffer,
UInt32 inBufferLength, UInt8 *outBuffer,
UInt32 *outBufferLength, UInt8 *signature,
UInt32 signatureLength,
VerifyResultType *verifyResult )

Parameters
-> libRef
CPM Library reference number (68k only).

-> keyInfo
An APKeyInfoStruct that represents the certificate’s encryption key. Extracting the key data from the certificate and constructing the APKeyInfoStruct (through CPMLibImportKeyInfo) is the caller’s responsibility.
-> verifyInfo  A pointer to an **APVerifyInfoStruct** that specifies the hash and cipher operations that should be performed during verification. This information is embedded as **APHashInfoStruct** and **APCipherInfoStruct** structures. If you want to use the default operations, allocate and zero the embedded structures. When the function returns, the structures will be populated with information describing the operations that were used.

-> inBuffer   A pointer to the message data.

-> inBufferLength  

<- outBuffer  A pointer to the buffer where the decrypted signature will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate all of the decrypted data. If you don’t care about the signature, pass NULL.

<- outBufferLength  

If outBuffer is NULL, set this to 0. Otherwise, you pass in the (allocated) length of outBuffer, in bytes. The function resets the argument to the amount of data that was actually copied into outBuffer. If the function returns cpmErrBufTooSmall, outBufferLength is set to the minimum buffer size that’s needed to accommodate the verified data.

-> signature  A pointer to the message’s signature. Extracting the signature from the message is the caller’s responsibility

-> signatureLength  

The length of the signature, in bytes.
<- verifyResult

An integer point that returns, by reference, the result of the verification. 0 means the message was verified; non-zero means it wasn’t. The meaning of a non-zero return is defined by the algorithm provider.

### Result

The function returns `errNone` upon success. For other error codes, see [CPM Error Codes](#).

### Comments

This function performs a block verification operation. For stream verification, use [CPMLibVerifyInit](#).

Keep in mind that a direct return of `errNone` doesn’t mean that the message has been verified. The verification status is returned by reference in `verifyResult`.

---

**New**

### CPMLibVerifyFinal

#### Purpose

Finalizes a verify session.

#### Declared In

CPMLib.h

#### Prototype

```c
Err CPMLibVerifyFinal ( UInt16 libRef,
                        APKeyInfoType *keyInfo,
                        APVerifyInfoType *verifyInfo, UInt8 *inBuffer,
                        UInt32 inBufferLength, UInt8 *outBuffer,
                        UInt32 *outBufferLength, UInt8 *signature,
                        UInt32 signatureLength,
                        VerifyResultType *verifyResult )
```

#### Parameters

- `libRef` CPM Library reference number (68k only).
- `verifyInfo` A pointer to the `APVerifyInfoStruct` that was returned by [CPMLibVerifyInit](#).
- `inBuffer` A pointer to the final buffer of message data, or `NULL` if there’s no more data.
-> inBufferLength
    The length of inBuffer, in bytes.

<- outBuffer
    A pointer to the buffer where the decrypted signature will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate all of the decrypted data. If you don’t care about the signature, pass NULL.

<- outBufferLength
    If outBuffer is NULL, set this to 0. Otherwise, you pass in the (allocated) length of outBuffer, in bytes. The function resets the argument to the amount of data that was actually copied into outBuffer. If the function returns cpmErrBufTooSmall, outBufferLength is set to the minimum buffer size that’s needed to accommodate the verified data.

-> signature
    A pointer to the message’s signature. Extracting the signature from the message is the caller’s responsibility.

-> signatureLength
    The length of the signature, in bytes.

<- verifyResult
    An integer point that returns, by reference, the result of the verification. 0 means the message was verified; non-zero means it wasn’t. The meaning of a non-zero return is defined by the algorithm provider.

Result
    The function returns errNone upon success. For other error codes, see CPM Error Codes.

Comments
    This function feeds a final (optional) buffer of message into a verify stream that was previously initialized by CPMLibVerifyInit and augmented through successive calls to CPMLibVerifyUpdate. The entire message is then verified, the results of the verification are returned in verifyResult, and the stream is closed.
New CPMLibVerifyInit

Purpose
Initiates a streaming verify operation.

Declared In
CPMLib68KInterface.h, CPMLibArmInterface.h

Prototype
Err CPMLibVerify ( UInt16 libRef,
APKeyInfoType *keyInfo,
APVerifyInfoType *verifyInfo )

Parameters
- libRef
CPM Library reference number (68k only).
- keyInfo
An APKeyInfoStruct that represents the certificate’s encryption key. Extracting the key data from the certificate and constructing the APKeyInfoStruct (through CPMLibImportKeyInfo) is the caller’s responsibility.
- verifyInfo
A pointer to an APVerifyInfoStruct that specifies the hash and cipher operations that should be performed during verification. This information is embedded as AHashInfoStruct and APCipherInfoStruct structures. If you want to use the default operations, allocate and zero the embedded structures. When the function returns, the structures will be populated with information describing the operations that were used.

Result
The function returns errNone upon success. For other error codes, see CPM Error Codes.

Comments
This function initializes a streaming hash operation. To feed data to the stream, you call CPMLibVerifyUpdate followed by CPMLibVerifyFinal. The “update” function is optional; the “final” function is mandatory. For a block hash, see CPMLibVerify.
New

CPMLibVerifyUpdate

Purpose
Sends message data to a streaming verify operation.

Declared In
CPMLib68KInterface.h, CPMLibArmInterface.h

Prototype
Err CPMLibVerifyUpdate ( UInt16 libRef,
APKeyInfoType *keyInfo,
APVerifyInfoType *verifyInfo )
UInt8 *inBuffer, UInt32 inBufferLength)

Parameters
- libRef CPM Library reference number (68k only).
- keyInfo The APKeyInfoStruct that was used in the CPMLibVerifyInit call.
- verifyInfo A pointer to the APVerifyInfoStruct that was returned by CPMLibVerifyInit.
- inBuffer A pointer to the message data.
- inBufferLength The length of inBuffer, in bytes.

Result
The function returns errNone upon success. For other error codes, see CPM Error Codes.

Comments
This function feeds message data into the verify stream that was intialized by CPMLibVerifyInit. When you’re finished feeding data into the stream, call CPMLibVerifyFinal to close the stream and return the verification results.

CPM Error Codes
The table below lists and explains the error codes that are returned by the CPM functions.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpmErrAlreadyOpen</td>
<td>Returned by CPMLibOpen() to indicate that the CPM library has already been opened by your application. The open library remains open.</td>
</tr>
<tr>
<td>cpmErrNotOpen</td>
<td>All CPM functions (except CPMLibOpen()) expect the CPM library to be open. This code is returned if the library isn’t open. To open the library, call CPMLibOpen().</td>
</tr>
<tr>
<td>cpmErrStillOpen</td>
<td>Returned by CPMLibClose() if the function was unable to close the library.</td>
</tr>
<tr>
<td>cpmErrNoProviders</td>
<td>Returned by CPMLibOpen() if the function was unable to locate (or load) any cryptography providers. Without a cryptography provider, none of the other CPM functions will work.</td>
</tr>
<tr>
<td>cpmErrNoBaseProvider</td>
<td>Returned by CPMLibOpen() if the function was unable to locate (or load) the default cryptography provider. (Note that this means that at least one non-default provider was found. If no providers were found, the function would have returned cpmErrNoProviders).</td>
</tr>
<tr>
<td>cpmErrProviderNotFound</td>
<td>Returned by CPMLibGenerateKey(), CPMLibEncrypt(), CPMLibDecrypt() and other algorithm-dependent functions if the requested provider couldn’t be found.</td>
</tr>
<tr>
<td>cpmErrParamErr</td>
<td>Returned by a number of CPM functions when an argument is invalid (a pointer that points to an unallocated structure; a structure field that isn’t properly set, and so on).</td>
</tr>
<tr>
<td>cpmErrOutOfMemory</td>
<td>Returned by CPMLibOpen() if the memory for a new library handle (and the structures it represents) couldn’t be allocated.</td>
</tr>
</tbody>
</table>
### Cryptography Provider Manager

#### CPM Error Codes

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpmErrBufTooSmall</td>
<td>Returned by CPMLibExportKey() and CPMLibExportContext() functions if the storage buffer (allocated by the caller) isn’t big enough to accommodate the key or context.</td>
</tr>
<tr>
<td>cpmErrBadData</td>
<td>Returned by algorithm functions (CPMLibEncrypt...(), CPMLibDecrypt...(), etc.) when the key or other required data is invalid.</td>
</tr>
<tr>
<td>cpmErrUnimplemented</td>
<td>Returned by functions that aren’t currently implemented.</td>
</tr>
<tr>
<td>cpmErrUnsupported</td>
<td>Returned by functions that aren’t currently supported.</td>
</tr>
<tr>
<td>cpmErrNoGlobals</td>
<td>Returned by functions that access global CPM data—functions such as CPMLibEnumerateProviders() and CPMLibGetProviderInfo()—when that data doesn’t exist.</td>
</tr>
<tr>
<td>cpmErrKeyExists</td>
<td>Returned by CPMLibImportKey() when the key you’re trying to import already exists.</td>
</tr>
<tr>
<td>cpmErrKeyNotFound</td>
<td>Returned by CPMLibExportKey() when the key you’re trying to export doesn’t exists.</td>
</tr>
</tbody>
</table>
SSL Functions

This chapter describes the functions that are defined in the SSL library. These functions let you create and configure an SSL context, apply the SSL protocol to an existing socket, and send SSL data to and receive data over the socket.

But before you do any of that, you have to load and open the SSL library; something like this:

```c
Err error;
UInt16 libRef;

if ( SyLibFind( kSslDBName, &libRef ) != 0 )
{
    error = SysLibLoad(kSslLibType, kSslLibCreator, &libRef);
}
/* error checking goes here. */
error = SslLibOpen( libRef );
...
```

After that, you typically...
1. ...create an `SslLib` through `SslLibCreate`,
2. spawn an `SslContext` through `SslContextCreate`,
3. set the context’s socket (see the `SslContextSet_Socket` macro),
4. “open” the context through `SslOpen` (this enables the SSL protocol),
5. send and receive data through `SslRead` and `SslWrite`,
6. and then close everything down (`SsslClose`, `SsslContextDestroy`, `SsslLibDestroy`, and `SsslLibClose`, in that order).

You can also use the more detailed `SsslSend` and `SsslReceive` functions to send and receive data. If you want to perform “streaming” reads—in which partial SSL records are read as the
SSL Functions
SSL Attribute Functions and Macros

...data arrives—use SslPeek and SslConsume. (For more on streaming, see the ReadStreaming attribute).

All functions described below are defined in SslLib.h.

SSL Attribute Functions and Macros

An SSL context is defined, primarily, by the values of its SSL attributes. These attributes are set and retrieved through attribute-specific macros. For example, the SslLibSet_InfoCallback macro sets an SslLib’s InfoCallback attribute.

The attribute macros are defined in terms of a set of eight functions listed in this chapter. These functions—SslLibSetLong, SslContextSetLong, SslLibSetPtr, SslContextSetPtr, and so on—can be called directly, but it’s suggested that you stick with the macros. If you want to call an attribute function directly, you have to identify the attribute by passing in one of the attribute constants.

The attribute macros and constants are described in Chapter 82, “SSL Attributes and Macros,” on page 2181.

A Note on the Function Names

All of the SSL library functions have “Ssl” as a prefix. Furthermore, the expected Palm library functions (SslLibOpen, SslLibClose) have an equally expected “SslLib” prefix. Unfortunately, another set of functions also uses “SslLib” as a prefix. These functions, SslLibCreate, SslLibDestroy, and so on, operate on instances of the SslLib data type, which type represents a generic SSL context.

Admittedly, “SslLib” is bad choice for the name of this data type. As pointed out in Chapter 81, “SSL Structures and Data Types,” when you see the SslLib data type, you should think “generic SSL context,” not “SSL library.”
SSL Library Functions

New SslClose

Purpose
Shuts down an SSL session.

Prototype
Err SslClose ( UInt16 libRef, SslContext *context,
UInt16 flags, UInt32 timeout )

Parameters
- libRef (68k only) SSL library reference number.
- context The context that you want to close.
- flags Options that set the session’s “shutdown” attributes.
- timeout Amount of time to wait for the final handshake messages from the server, in milliseconds.

Result
errNone means success; for other codes, see Chapter 83, “SSL Error Codes.”

Comments
By default, an SSL shutdown involves an exchange of messages with the server. As with SslOpen, if this function times out, simply call it repeatedly until you get confirmation that the session has actually been closed (i.e. until the function returns errNone). Unlike with SslClose, you needn’t clear the flags when you re-call this function.
The flags set the SSL attributes that are used during the shutdown:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>sslCloseUseDefaultTimeout</td>
<td>If this flag is set, the current value of the IoTimeout attribute is used as the function’s timeout, overriding the timeout argument. If it isn’t set, IoTimeout is set to timeout and is used as the timeout for this function.</td>
</tr>
<tr>
<td>sslCloseDontSendShutdown</td>
<td>Sets the DontSendShutdown attribute to 1. This suppresses the shutdown messages.</td>
</tr>
<tr>
<td>sslCloseDontWaitForShutdown</td>
<td>Sets the DontWaitForShutdown attribute to 1. This tells the function to return after it sends a shutdown message to the server, thus ignoring the server’s response.</td>
</tr>
</tbody>
</table>

---

**New**

**SslConsume**

**Purpose**
Removes data from a context’s in-coming data buffer. Use after an SslPeek only.

**Prototype**
```c
void SslConsume ( UInt16 libRef,
                 SslContext *context, Int32 availableBytes )
```

**Parameters**
- `-> libRef` (68k only) SSL library reference number.
- `-> context` Context you want to look at.
- `-> availableBytes` The number of bytes to remove. This should always be the value that’s returned to you by SslPeek’s availableBytes argument.

**Comments**
You call this function after calling SslPeek to remove the peeked at data.
New

SslContextCreate

Purpose
Creates a new SSL context. The object is used to open, read, write, and close an SSL session.

Prototype
Err SslContextCreate ( UInt16 libRef, SslLib *contextTemplate, SslContext **context )

Parameters
- libRef (68k only) SSL library reference number.
- contextTemplate
  Pointer to an SslLib object that will be used as the template for this context.
- context
  You pass in the address of an SslContext pointer. The function allocates a new SslContext object and points your pointer at it.

Result
errNone means success; for other codes, see Chapter 83, “SSL Error Codes.”

Comments
The attributes that have been set in contextTemplate are copied into context. These attributes don’t include a network socket; setting the socket is typically the first thing you do with your context object. For instructions, see Socket in Chapter 82, “SSL Attributes and Macros,” on page 2181.

After you set its socket, the context object can be used to open an SSL session; see SslOpen.

When you’re finished using your context object, you close the session (SsslClose) and the destroy the object by passing it to SsslContextDestroy.
**SSL Functions**

**SSL Library Functions**

---

**New**

**SslContextDestroy**

**Purpose**

Destroys an SSL context. You should close the object’s SSL session (see `SslClose`) before destroying it.

**Prototype**

```c
Err SslContextDestroy ( Uint16 libRef, SslContext *context)
```

**Parameters**

- `- `libRef` (68k only) SSL library reference number.
- `- `context` The context you wish to demolish.

**Result**

`errNone` means success; for other codes, see Chapter 83, “SSL Error Codes.”

**See Also**

`SslContextCreate`

---

**New**

**SslContextGetLong**

**Purpose**

Returns the value of an integer-valued SSL attribute retrieved from a context.

**Prototype**

```c
Int32 SslLibGetLong ( Uint16 libRef, SslContext *context, SslAttribute attribute )
```

**Parameters**

- `- `libRef` (68k only) SSL library reference number.
- `- `context` The context that contains the attribute.
- `- `attribute` Constant that represents the attribute that you want the value of. See “SSL Attribute Constants” in Chapter 82, “SSL Attributes and Macros,” on page 2181 for a list of attribute constants.

**Result**

The attribute’s value is returned directly.
Comments

You should rarely need to invoke this function directly. Instead, use the attribute macros, as described in “SSL Attribute Functions and Macros” near the beginning of this chapter.

See Also

Chapter 82, “SSL Attributes and Macros,” on page 2181,
SslContextSetLong, SslContextSetPtr, SslContextGetPtr

New

SslContextGetPtr

Purpose

Returns a pointer to an attribute that’s owned by a context template.

Prototype

Err SslLibGetPtr ( UInt16 libRef,
                  SslContext *context, SslAttribute attribute,
                  void **value )

Parameters

- `libRef` (68k only) SSL library reference number.
- `context` The context that contains the attribute.
- `attribute` Constant that represents the attribute that you want the value of. “SSL Attribute Constants” in Chapter 82, “SSL Attributes and Macros,” on page 2181 for a list of attribute constants.
- `value` You pass in the address of a pointer; the function will point your pointer to the attribute’s data. The data’s type depends on the attribute. You mustn’t free or modify the pointed to data.

Result

errNone means success; for other codes, see Chapter 83, “SSL Error Codes.”
SSL Functions
SSL Library Functions

New SslContextSetLong

Purpose
Modifies a context by changing the value of one of its integer-valued SSL attributes.

Prototype
Err SslLibSetLong ( UInt16 libRef,
             SslContext *context, SslAttribute attribute,
             Int32 value )

Parameters
-> libRef (68k only) SSL library reference number.
-> context The context template you want to modify.
-> attribute Constant that represents the attribute that you want to change. See “SSL Attribute Constants” in Chapter 82, “SSL Attributes and Macros,” on page 2181 for a list of attribute constants.
-> value The attribute’s new (desired) value.

Result
errNone means success; for other codes, see Chapter 83, “SSL Error Codes.”

Comments
You should rarely need to invoke this function directly. Instead, use the attribute macros, as described in “SSL Attribute Functions and Macros” near the beginning of this chapter.

See Also
Chapter 82, “SSL Attributes and Macros,” on page 2181,
SslContextSetPtr, SslContextGetLong,
SslContextGetPtr
SslContextSetPtr

Purpose
Modifies a context template by changing the value of one of its pointer-valued SSL attributes.

Prototype
Err SslLibSetPtr ( Uint16 libRef,
SslContext *context, SslAttribute attribute,
void *value )

Parameters
-> libRef (68k only) SSL library reference number.
-> context The context you want to modify.
-> attribute Constant that represents the attribute that you want to change. See “SSL Attribute Constants” in Chapter 82, “SSL Attributes and Macros,” on page 2181 for a list of attribute constants.
-> value A pointer to the attribute’s new (desired) value. The type of data that the pointer should point to is defined by the attribute.

Result
errNone means success; for other codes, see Chapter 83, “SSL Error Codes.”

Comments
You should rarely need to invoke this function directly. Instead, use the attribute macros, as described in “SSL Attribute Functions and Macros” near the beginning of this chapter.

See Also
Chapter 82, “SSL Attributes and Macros,” on page 2181, SslContextSetLong, SslContextGetLong, SslContextGetPtr
New SslFlush

Purpose    Flushed a context’s out-going data buffer, sending the data to the network.

Prototype  Err SslFlush ( Int16 libRef, SslContext *context, Int32 *outstanding )

Parameters -> libRef       (68k only) SSL library reference number.
              -> context     Context you want to flush.
              <- outstanding    The number of bytes of data left in the buffer.

Result     errNone means success; for other codes, see Chapter 83, “SSL Error Codes.”

Comments   If the context is in autoflush mode (the AutoFlush attribute is 0), SslWrite and SslSend calls will write their data into the context’s out-going data buffer, but the data won’t actually be sent to the network. To “flush” this data, you have to explicitly call SslFlush.

By not autoflushing, you can make multiple, small SslWrite/ SslSend followed by an SslFlush and thereby improve network efficiency. Although the out-going buffer size is set to the value of the WbufSize attribute, the actual number of bytes that can be written by your application is somewhat smaller due to SSL overhead. After performing a write, you should check the value of the WriteBufPending attribute; if it’s approaching the WbufSize value, you should call SslFlush.

When the function returns, the outstanding value tells how many bytes of data are left in the buffer. If this value is non-zero, the next SslWrite, SslSend, or SslFlush call will attempt to write those bytes to the network.
New

SslLibClose

Purpose
Closes a reference to the SSL library.

Prototype
Err SslLibClose ( UInt16 refNum )

Parameters
- refNum
SSL library reference number (68k only).

Result
errNone means success; for other codes, see Chapter 83, “SSL Error Codes.”

Comments
You can only a library that you’ve already opened. To open the SSL library, call SslLibOpen.

New

SslLibCreate

Purpose
Allocates and returns new SslLib object. The object is a template that describes a generic SSL context, and is used to create “real” context objects (SslContext).

Declared In
SslLib.h

Prototype
Err SslLibCreate ( UInt16 libRef, SslLib **contextTemplate )

Parameters
- libRef
(68k only) SSL library reference number.

- contextTemplate
You pass in the address of an SslLib pointer. The function allocates an SslLib object for you, and points your pointer at it.

Result
errNone means success; for other codes, see Chapter 83, “SSL Error Codes.”
SSL Functions
SSL Library Functions

Comments
You modify the context template (i.e. the SslLib object) by calling the SSL attribute macros described in Chapter 82, “SSL Attributes and Macros.” You then use it as a template with which you create SslContext objects (see SslContextCreate). The latter objects are needed for actual SSL data transaction operations.

The contextTemplate that’s returned by this function should ultimately be destroyed through SslLibDestroy.

For more on SslLib, SslContext, and the relationship between them, see their descriptions in Chapter 81, “SSL Structures and Data Types,” on page 2163.

New

SslLibDestroy

Purpose
Destroys an SslLib object and frees all the memory it allocated.

Prototype
void SslLibDestroy ( UInt16 libRef, SslLib *contextTemplate )

Parameters
-> libRef     (68k only) SSL library reference number.
-> contextTemplate     The SslLib you want to destroy, as previously returned by SslLibCreate.

Result
errNone means success; for other codes, see Chapter 83, “SSL Error Codes.”

Comments
The SslContext objects that were spawned by this SslLib maintain a reference to their spawner. Because of this, you shouldn’t destroy an SslLib until you’ve destroyed all of its contexts.
New SslLibOpen

Purpose
Opens a reference to the SSL library.

Prototype
Err SslLibOpen ( UInt16 libRef )

Parameters
-> libRef (68k only) SSL library reference number.

Result
errNone means success; for other codes, see Chapter 83, “SSL Error Codes.”

Comments
When you’re done using the SSL library, you must close it through SslLibClose.

New SslLibGetLong

Purpose
Returns the value of an integer-valued SSL attribute retrieved from a context template.

Prototype
Int32 SslLibGetLong ( UInt16 libRef, SslLib *contextTemplate, SslAttribute attribute )

Parameters
-> libRef (68k only) SSL library reference number.
-> contextTemplate The context template that contains the attribute.
-> attribute Constant that represents the attribute that you want the value of. See “SSL Attribute Constants” in Chapter 82, “SSL Attributes and Macros,” on page 2181 for a list of attribute constants.

Result
The attribute’s value is returned directly.
Comments

You should rarely need to invoke this function directly. Instead, use the attribute macros, as described in “SSL Attribute Functions and Macros” near the beginning of this chapter.

See Also

Chapter 82, “SSL Attributes and Macros,” on page 2181, SslLibSetLong, SslLibSetPtr, SslLibGetPtr

New

SslLibGetPtr

Purpose

Returns a pointer to an attribute that’s owned by a context template.

Prototype

Err SslLibGetPtr ( UInt16 libRef, SslLib *contextTemplate, SslAttribute attribute, void **value )

Parameters

- `libRef` (68k only) SSL library reference number.
- `contextTemplate` The context template that contains the attribute.
- `attribute` Constant that represents the attribute that you want the value of. “SSL Attribute Constants” in Chapter 82, “SSL Attributes and Macros,” on page 2181 for a list of attribute constants.
- `value` You pass in the address of a pointer; the function will point your pointer to the attribute’s data. The data’s type depends on the attribute. You mustn’t free or modify the pointed to data.

Result

errNone means success; for other codes, see Chapter 83, “SSL Error Codes.”
Comments
You should rarely need to invoke this function directly. Instead, use the attribute macros, as described in “SSL Attribute Functions and Macros” near the beginning of this chapter.

See Also Chapter 82, “SSL Attributes and Macros,” on page 2181, SslLibSetLong, SslLibGetLong, SslLibSetPtr

New SslLibSetLong

Purpose Modifies a context template by changing the value of one of its integer-valued SSL attributes.

Prototype Err SslLibSetLong ( UInt16 libRef, SslLib *contextTemplate, SslAttribute attribute, Int32 value )

Parameters
- > libRef (68k only) SSL library reference number.
- > contextTemplate The context template you want to modify.
- > attribute Constant that represents the attribute that you want to change. See “SSL Attribute Constants” in Chapter 82, “SSL Attributes and Macros,” on page 2181 for a list of attribute constants.
- > value The attribute’s new (desired) value.

Result errNone means success; for other codes, see Chapter 83, “SSL Error Codes.”

Comments You should rarely need to invoke this function directly. Instead, use the attribute macros, as described in “SSL Attribute Functions and Macros” near the beginning of this chapter.

See Also Chapter 82, “SSL Attributes and Macros,” on page 2181, SslLibSetPtr, SslLibGetLong, SslLibGetPtr
New  SsslLibSetPtr

**Purpose**
Modifies a context template by changing the value of one of its
pointer-valued SSL attributes.

**Prototype**
```
Err SsslLibSetPtr ( UInt16 libRef,
                  SsslLib *contextTemplate, SsslAttribute attribute,
                  void *value )
```

**Parameters**
- `-> libRef`  
(68k only) SSL library reference number.
- `-> contextTemplate`  
The context template you want to modify.
- `-> attribute`  
Constant that represents the attribute that you
want to change. See “SSL Attribute Constants”
in Chapter 82, “SSL Attributes and Macros,” on
page 2181 for a list of attribute constants.
- `-> value`  
A pointer to the attribute’s new (desired) value. 
The type of data that the pointer should point
to is defined by the attribute.

**Result**
errNone means success; for other codes, see Chapter 83, “SSL Error
Codes.”

**Comments**
You should rarely need to invoke this function directly. Instead, use
the attribute macros, as described in “SSL Attribute Functions and
Macros” near the beginning of this chapter.

**See Also**
Chapter 82, “SSL Attributes and Macros,” on page 2181,
SsslLibSetLong, SsslLibGetLong, SsslLibGetPtr
**New** **SslOpen**

**Purpose** Opens an SSL session, possibly as a continuation of a previous session.

**Prototype**
```c
Err SslOpen ( UInt16 libRef, SslContext *context, UInt16 flags, UInt32 timeout )
```

**Parameters**
- `-> libRef` (68k only) SSL library reference number.
- `-> context` The context object that will configure and own the session.
- `-> flags` Options that set various SSL attributes. The most important flags are `sslOpenModeSsl` and `sslOpenModeClear`. These mutually exclusive flags tell the session to use SSL or cleartext, respectively.
- `-> timeout` Amount of time to wait for the handshake confirmation before giving up, in milliseconds.

**Result** `errNone` means success; for other codes, see Chapter 83, “SSL Error Codes.”

**Comments** Calling `SslOpen` is essentially the same as setting the attributes that correspond to the function’s flags:

You should rarely need to invoke this function directly. Instead, use the attribute macros, as described in “SSL Attribute Functions and Macros” near the beginning of this chapter.

**See Also** Chapter 82, “SSL Attributes and Macros,” on page 2181, `SslContextGetLong`, `SslContextSetLong`, `SslContextSetPtr`
## SSL Functions

**SSL Library Functions**

If the function times out, you should simply call it again—timing out isn’t a fatal error. Each time you call `SslOpen` (after a timeout), the handshake continues from where it timed out. However, it’s important that when you re-call `SslOpen`, that you clear the `flags` argument. The attributes that they affect will already have been set; setting them a second time could cause the context to be reset, which will cause the handshake to start from the beginning.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sslOpenModeClear</code></td>
<td>Sets the context’s <code>Mode</code> attribute to <code>sslModeClear</code> (cleartext).</td>
</tr>
<tr>
<td><code>sslOpenModeSsl</code></td>
<td>Sets the context’s <code>Mode</code> attribute to <code>sslModeSslClient</code> (SSL is enabled). This overrides <code>sslOpenModeClear</code>.</td>
</tr>
<tr>
<td><code>sslOpenNoAutoFlush</code></td>
<td>Sets the <code>AutoFlush</code> attribute to 0. This suppresses auto-flushing so that <code>SslWrite/SslSend</code> data is cached until you call <code>SslFlush</code>.</td>
</tr>
<tr>
<td><code>sslOpenNewConnection</code></td>
<td>Sets the context’s <code>SslSession</code> attribute to <code>NULL</code>. This will force a full SSL handshake. If you don’t set this flag, the previous <code>SslSession</code> will be used (if possible), and the handshake is truncated.</td>
</tr>
<tr>
<td><code>sslOpenBufferedReuse</code></td>
<td>Sets the <code>BufferedReuse</code> attribute to 1. This provides an even greater savings during a (truncated) handshake.</td>
</tr>
<tr>
<td><code>sslOpenUseDefaultTimeout</code></td>
<td>If this flag is set, the current value of the <code>IoTimeout</code> attribute is used as the function’s timeout, overriding the <code>timeout</code> argument. If it isn’t set, <code>IoTimeout</code> is set to <code>timeout</code> and is used as the timeout for this function.</td>
</tr>
<tr>
<td><code>sslOpenDelayHandshake</code></td>
<td>Instead of sending the final handshake message, cache it until the write buffer is flushed (through an autoflush, or through an <code>SslFlush</code> call). There is no attribute that corresponds to this flag.</td>
</tr>
</tbody>
</table>

If the function times out, you should simply call it again—timing out isn’t a fatal error. Each time you call `SslOpen` (after a timeout), the handshake continues from where it timed out. However, it’s important that when you re-call `SslOpen`, that you clear the `flags` argument. The attributes that they affect will already have been set; setting them a second time could cause the context to be reset, which will cause the handshake to start from the beginning.
**New**

**SslPeek**

**Purpose**
Lets you look at in-coming data without actually “consuming” it.

**Prototype**
```
Err SslPeek ( UInt16 libRef, SslContext *context, 
             void **data, Int32 *availableBytes, 
             Int32 maxAvailable)
```

**Parameters**
- `-> libRef` (68k only) SSL library reference number.
- `-> context` Context you want to look at.
- `<- data` Pass in the address of a pointer; the function sets your pointer to point to the context’s in-coming data buffer.
- `<- availableBytes` The function returns the smaller of a) the number of bytes of data that are waiting to be read or b) the value of `maxAvailable`.
- `-> maxAvailable` Sets a limit on the `availableBytes` value.

**Result**
Always returns 0.

**Comments**
This function is similar to `SslRead`, but rather than copy data into your buffer, it gives you direct access to the context’s in-coming data buffer. If the buffer is empty, data is read from the network until there are bytes available. After each successful `SslPeek` call, you should call `SslConsume` to clear the “peeked at” data. If you don’t consume after a peek, the next `SslPeek` will return a pointer to the same data.

`SslPeek` is meant to be used in contexts that have the `ReadStreaming` attribute set.
New SslRead

Purpose
Copies a buffer of in-coming data.

Prototype
Int16 SsslRead ( UInt16 libRef,
SslContext *context, void *buffer,
UInt16 bufferLength, Err *error )

Parameters
- `libRef` (68k only) SSL library reference number.
- `context` Context that owns the session.
- `buffer` This is where the retrieved data will be copied.
The buffer must be allocated by the caller.
- `bufferLength` The length of `buffer`, in bytes.
- `error` Error code as listed in see Chapter 83, “SSL Error Codes.”

Result
Returns the number of bytes that were actually read. A return of 0 means the socket was shut down by the server. If the return is -1, look in `error` for the precise error code.

Comments
This is a convenient cover for `SsslReceive`. It ignores the address of the sender, doesn’t set any socket flags, and uses the context’s current timeout setting (the `IoTimeout` attribute).
SSL Functions
SSL Library Functions

\v

\textbf{New} SslReceive

\textbf{Purpose} Copies a buffer of in-coming data.

\textbf{Prototype} \begin{verbatim}
Int16 SsslReceive ( UInt16 libRef, SsslContext *context, void *buffer, UInt16 bufferLength, UInt16 flags, void *fromAddress, UInt16 *fromAddressLength, Int32 timeout, Err *error )
\end{verbatim}

\textbf{Parameters} 
- \texttt{libRef} (68k only) SSL library reference number.
- \texttt{context} Context that owns the session.
- \texttt{buffer} This is where the retrieved data will be placed.
  \begin{itemize}
  \item \texttt{bufferLength} The length of \texttt{buffer}, in bytes.
  \item \texttt{flags} Options that are applied to (and stored by) the socket. See \texttt{I/O Flags} in Chapter 61, “Net Library,” on page 1413 for a description of these options.
  \item \texttt{fromAddress} Optionally returns the address of the sender. The \texttt{fromAddress} buffer must be allocated by the caller. If you don’t want to retrieve the address, pass a \texttt{NULL} pointer.
  \item \texttt{fromAddressLength} On input, you set \texttt{*fromAddressLength} to the size of the \texttt{fromAddress} buffer, in bytes. Upon return, \texttt{*fromAddressLength} is set to the actual size of the buffer. If you don’t want to retrieve the address, pass a \texttt{NULL} pointer.
  \item \texttt{timeout} Maximum timeout in system ticks; \texttt{-1} means wait forever.
  \end{itemize}
SSL Functions
SSL Library Functions

Result
Returns the number of bytes that were actually read. A return of 0 means the socket was shut down by the server. If the return is -1, look in error for the precise error code.

Comments
Unlike in SslSend, the address that you supply (fromAddress) doesn’t overwrite the socket’s address. This is a cover for the Network library’s NetLibDmReceive function.

For a shorthand version of this function, see SslRead.

New SslSend

Purpose
Writes a buffer of data to the network, or buffers it for a later send.

Prototype
Int16 SslSend ( UInt16 libRef,
SslContext *context, void *buffer,
UInt16 bufferLength, UInt16 flags,
void *toAddress, UInt16 toAddressLength,
Int32 timeout, Err *error )

Parameters
- > libRef (68k only) SSL library reference number.
- > context Context that owns the SSL session.
- > buffer A pointer to the data that you want to write.
- > bufferLength The amount of data you want to write, in bytes.
- > flags Options that are applied to (and stored by) the socket. See I/O Flags in Chapter 61, “Net Library,” on page 1413 for a description of these options.
- > toAddress Address of the recipient. If this is a NetSocketAddrType, the socket’s address is set to this value. If it’s 0, the socket’s current address is used.
SSL Functions

SSL Library Functions

-> toAddressLength
   Size of *toAddress, in bytes.

-> timeout
   Maximum timeout in system ticks; -1 means wait forever.

<- error
   Error code as listed in see Chapter 83, “SSL Error Codes.”

Result
Returns the number of bytes that were actually sent. A return of 0 means the socket was shut down by the server. If the return is -1, look in error for the precise error code.

Comments
If the context has the AutoFlush attribute enabled, the data is immediately written to the network; otherwise, it’s cached until SslFlush is called. This function is a cover for the Network library’s NetLibSend function.

For a shorthand version of this function, see SslWrite.

New
SslWrite

Purpose
Writes a buffer of data to the network, or caches it in anticipation of a flush.

Prototype
Int16 SslWrite ( UInt16 libRef,
SslContext *context, void *buffer,
UInt16 bufferLength, Err *error )

Parameters
-> libRef
   (68k only) SSL library reference number.

-> context
   Context that owns the session.

-> buffer
   A pointer to the data that you want to write.

-> bufferLength
   The amount of data you want to write, in bytes.
SSL Functions
Application-Defined Functions

Error code as listed in see Chapter 83, “SSL Error Codes.”

Result
Returns the number of bytes that were actually sent. A return of 0 means the socket was shut down by the server. If the return is -1, look in error for the precise error code.

Comments
This is a convenient cover for SslSend. It doesn’t set any socket flags, uses the socket’s current address, and uses the context’s current timeout setting (the IoTimeout attribute).

Application-Defined Functions

New SslCallbackFunc

Purpose
Prototype for SSL information and verification callback functions.

Prototype
typedef Int32 (*SslCallbackFunc)(SslCallback *callbackStruct, Int32 command, Int32 flavor, void *info);

Parameters
- > callbackStruct
  A structure that contains information about this callback function, including application-defined data that can be used in the function’s implementation. See SslCallback in Chapter 81, “SSL Structures and Data Types,” on page 2163.

- > command
  A constant that represents the general reason for the function’s invocation.

- > flavor
  A constant that refines the command value.
SSL Functions
Application-Defined Functions

-> info Additional data that may be needed by the callback. The type of data that’s passed depends on the command/flavor values.

Result The function should return errNone (0) upon success, otherwise, it should return one of the error codes described in Chapter 83, “SSL Error Codes.” A non-zero return value here is returned to the SSL function that caused the callback to be invoked.

Comments For every context, you can install two callback functions: An “info” callback and a “verify” callback:

- The info callback is typically used for debugging, displaying data, tracking progress, and so on. It’s invoked at well-defined junctures as the context is working.
- The verify callback is called when a certificate is being verified. It’s expected to handle error situations.

To install a callback, you create an SslCallback structure, populate the necessary fields, and then use it to set the value of the InfoCallback or VerifyCallback attribute. When you set the info callback, you also have to set the InfoInterest attribute to register for specific events (which will show up in the flavor argument when your callback is invoked). The verify callback doesn’t require similar event registration.

Setting the info callback in an SslContext looks a little bit like this:

```c
/* Create an SslCallback structure, and point its
 * ‘callback’ field to your function (‘MyInfoCallbackFunc’).
 */
SslCallback infoCallback;
infoCallback.callback = MyInfoCallbackFunc;

/* Set the InfoCallback attribute. */
SslContextSet_InfoCallback( sslContext, &infoCallback );

/* Set the events you’re interested in hearing about. */
SslContextSet_InfoInterest( sslContext,
                          sslFlgInfoAlert | sslFlgInfoHandshake | sslFlgInfoIo);
```

When you set the InfoCallback or VerifyCallback attribute, the SslCallback structure is copied into the object. Furthermore,
when you spawn an `SslContext` from an `SslLib`, the callback structures are copied into the new object.

**Commands and Flavors**

Taken together, the command and flavor arguments tell a callback why it’s being called. There are three command groups:

- The common command values (`sslCmdNew`, `sslResetNew`, and `sslCmdFree`) apply to both info and verify callbacks. These commands don’t use flavor arguments.
- `sslCmdInfo` is used for info callbacks only. The flavor argument indicates the reason for the callback.
- Similarly, `sslCmdVerify` is used for verify callbacks only (and flavor refines the command’s meaning).

**Common Commands**

The common commands are:

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sslCmdNew</code></td>
<td>The callback function was just copied into an object. (More accurately, the <code>SslCallback</code> that contains the function was copied.) This happens when you set the <code>InfoCallback</code> or <code>VerifyCallback</code> attribute, or when you spawn an <code>SslContext</code>.</td>
</tr>
<tr>
<td><code>sslCmdReset</code></td>
<td>The context that contains this callback was just reset (<code>SslContext</code> only; see the <code>Mode</code> attribute for a brief explanation of the SSL reset).</td>
</tr>
<tr>
<td><code>sslCmdFree</code></td>
<td>The <code>SslCallback</code> structure is about to be freed. This usually means that the <code>SslLib</code> or <code>SslContext</code> that owns the structure is being destroyed.</td>
</tr>
</tbody>
</table>

When a callback function is invoked with one of these common commands, the flavor argument is 0 (i.e. undefined) and info is NULL.
### sslCmdInfo Flavors

As mentioned above, the `sslCmdInfo` command argument is only sent to the info callback (keep in mind, however, that the info callback may also receive any of the common commands).

The flavor values that are associated with the `sslCmdInfo` command are:

<table>
<thead>
<tr>
<th>Flavor</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sslArgInfoHandshake</code></td>
<td>Notification of an SSL state change. Read the <code>HsState</code> attribute to determine the new state. The callback’s <code>info</code> parameter is set to <code>NULL</code>.</td>
</tr>
<tr>
<td><code>sslArgInfoAlert</code></td>
<td>Notification of an SSL alert. Your callback can read the <code>LastAlert</code> attribute to determine which alert was received. The callback’s <code>info</code> parameter is set to <code>NULL</code>.</td>
</tr>
<tr>
<td><code>sslArgInfoReadBefore</code></td>
<td>Notifications that are sent just before and just after data is read from or written to the network.</td>
</tr>
<tr>
<td><code>sslArgInfoReadAfter</code></td>
<td>The <code>info</code> argument is an <code>SslIoBuf</code> structure that contains the data.</td>
</tr>
<tr>
<td><code>sslArgInfoWriteBefore</code></td>
<td></td>
</tr>
<tr>
<td><code>sslArgInfoWriteAfter</code></td>
<td></td>
</tr>
<tr>
<td><code>sslArgCert</code></td>
<td>Notification that the server’s certificate chain has been verified. The <code>info</code> argument is an <code>SslExtendedItems</code> structure that contains the server’s certificate.</td>
</tr>
</tbody>
</table>

### sslCmdVerify Flavors

The verify callback function is called with the `sslCmdVerify` command when an error occurs during certificate verification, and
when the certificate is successfully verified. The associated flavor values are:

<table>
<thead>
<tr>
<th>Flavor</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>sslErrVerifyBadSignature</td>
<td>The certificate’s signature is invalid</td>
</tr>
<tr>
<td>sslErrVerifyNoTrustedRoot</td>
<td>A trusted certificate store (necessary for certificate verification) couldn’t be found</td>
</tr>
<tr>
<td>sslErrVerifyNotAfter</td>
<td>The certificate has expired.</td>
</tr>
<tr>
<td>sslErrVerifyNotBefore</td>
<td>The certificate is too early (the timestamp window is in the future).</td>
</tr>
<tr>
<td>sslErrVerifyConstraintViolation</td>
<td>The certificate violates an X509 extension</td>
</tr>
<tr>
<td>sslErrVerifyUnknownCriticalExtension</td>
<td>An X509 extension (that’s marked as “critical”) isn’t understood by the certificate verification routines.</td>
</tr>
<tr>
<td>sslErrOk</td>
<td>The certificate was successfully verified</td>
</tr>
</tbody>
</table>

Note that these constants are also used as error codes. If your verify function can’t fix the problem, it should return the command argument as an error.

In all cases, the info argument is an SslVerify structure that contains information about the certificate that’s being verified.
SSL Structures and Data Types

This chapter describes the structures and data types that are used by SSL.

All elements described below are defined in SslLib.h.

SSL Data Types

New SslAttribute

The SslAttribute data type is used to cast the SSL attribute constants that are passed to the attribute-setting functions.

typedef UInt32 SslAttribute;

In general, you should use the attribute-setting macros rather than the functions. The attribute upon which a macro operates is embedded in its name, so you rarely have to deal with SslAttribute types.

For more information on the relationship between the attribute-setting functions and macros, see Chapter 80, “SSL Functions,” on page 2135. For a list of the macros themselves, go to Chapter 82, “SSL Attributes and Macros,” on page 2181.

New SsslContext

SsslContext is the data type for a private structure that holds all the information, or context, associated with the SSL protocol that will be used in an SSL session. It contains flags that govern how the
SSL protocol will operate, read and write buffers where SSL packets are assembled and disassembled, various structures that are created as part of the SSL handshake, and so on.

The data type is declared in SslLib.h as:

```c
typedef struct SslContext_st SslContext;
```

The lifetime of an SslContext follows this pattern:

1. **Creation.** You never allocate your own SslContext objects; instead, you first create an SslLib (which see) and pass it to the SslContextCreate function. The SslLib acts as a template that’s used by the function to create and configure a new SslContext. This “configuration” consists of copying the SslLib’s **SSL attributes** into the new SslContext.

2. **Configuration.** After you get your hands on the new SslContext, you can refine its SSL attributes by calling the SSL attribute macros.

3. **Socket specification.** The one essential attribute that you must set is the context’s socket. The template doesn’t contain a reference to a socket, so if you want to actually use the SslContext object’s that you create, you have to explicitly set its socket. See **Socket** in Chapter 82, “SSL Attributes and Macros.”

4. **SSL sessions.** You then pass the SslContext to SslOpen, which creates a new SSL session. The SslContext is used as a cookie in session operations such as SslRead and SslClose. When you’re done with the session, you call SslClose. The same SslContext can be reused in successive sessions, but only one at a time.

5. **Destruction.** When you’re done using the SSL context, you hand the object back to the system through SslContextDestroy.

---

**New**

**SslLib**

SslLib is the data type for a private structure that describes a generic SSL context. It’s declared in SslLib.h as:
typedef struct SslLib_st  SslLib;

NOTE:  SslLib is a misleading name for this data type. The type doesn’t represent the SSL library, nor does it serve any explicit library-related function. Where you see “SslLib” you should think “generic SSL context.”

After you create and modify an SslLib object, you use it as a template to create “real” SSL context objects (type SslContext). When you’re done with the SslLib, you ask the SSL library to destroy it.

Functionally, the lifetime of an SslLib follows this pattern:
1. **Creation.** You create the object through SslLibCreate.
2. **Modification.** Since the structure that the SslLib represents is private, you can’t touch its fields directly. Instead, you use a set of macros (defined in SslLibMac.h) to fine-tune the object. These macros set the values of the SSL attributes that the SslLib contains. The SSL attributes and the macros that you use to set and get their values are described in Chapter 82, “SSL Attributes and Macros,” on page 2181.
3. **Context creation.** When you’re satisfied with your SslLib’s attribute configuration, you use it to create new SslContext objects through the SslContextCreate function. The SslLib’s attributes are copied into the new SslContext. Subsequent changes to the SslLib won’t affect the SslContext objects that you’ve already created.
4. **Destruction.** When you’re done with your SslLib and all of its SslContexts you should destroy it by passing it to SslLibDestroy. Note that each SslContext object keeps a reference to the SslLib that calved it, so you mustn’t destroy an SslLib until after its SslContext objects have been freed.

You can create as many SslLib objects as you want, but you typically only create one per application.
SSL Structures and Data Types

SSL Structures

New

SslCallback

The SslCallback structure contains information about a callback function that’s invoked during SSL operations.

```c
typedef struct {
    void  *reserved;
    SslLibCallbackFunc  callback;
    void  *data;
    SslContext  *ssl;
}  SslCallback
```

The fields are:

<table>
<thead>
<tr>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>callback</td>
</tr>
<tr>
<td>data</td>
</tr>
<tr>
<td>ssl</td>
</tr>
<tr>
<td>reserved</td>
</tr>
</tbody>
</table>

There are two types of callback functions: info and verify. The info callback is called as data is being read, when the server sends an alert, and so on. The verify callback is called when a certificate is being verified.

To register a callback function, you create an SslLibCallback structure, fill out the callback and data fields, and then use the structure to set the value of the InfoCallback or
VerifyCallback attribute. To set the attribute you call one of these macros:

- `SslLibSet_InfoCallback` sets the InfoCallback attribute of an `SslLib` object.
- `SslLibSet_VerifyCallback` sets the VerifyCallback attribute of an `SslLib` object.
- `SslContextSet_InfoCallback` sets the InfoCallback attribute of an `SslContext` object.
- `SslContextSet_VerifyCallback` sets the VerifyCallback attribute of an `SslContext` object.

When you register an info callback you must also set the `InfoInterest` attribute by calling `SslLibSet_InfoInterest` or `SslContextSet_InfoInterest` macro. The `InfoInterest` attribute contains a list of the events that your info callback is interested in.

As with (nearly) all SSL attributes, the callbacks that you set in an `SslLib` are copied into the `SslContext` objects that it spawns.

You can use the same `SslLibCallback` structure to set more than one callback function. After you’ve called a callback-registering macro, you can free the original `SslLibCallback` structure.

The protocol for the callback functions is described in `SslCallbackFunc`, in Chapter 80, “SSL Functions.”

### New

**SslCipherSuiteInfo**

Structure that contains information about the cipher suite that’s being used. The structure is stored in the `CipherSuiteInfo` attribute.

```c
typedef struct SslCipherSuiteInfo_st {
    UInt8  cipherSuite[2];
    UInt16  cipher;
    UInt16  digest;
    UInt16  keyExchange;
    UInt16  authentication;
    UInt16  version;
} SslCipherSuiteInfo_st;
```
SSL Structures and Data Types

### SSL Structures

```c
UInt16  cipherBitLength;
UInt16  cipherKeyLength;
UInt16  keyExchangeLength;
UInt16  authenticationLength;
UInt16  export;
}

SslCipherSuiteInfo;
```

The fields are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cipherSuite</td>
<td>A two-byte value that represents the current cipher suite. The possible values, encoded as constants, are:</td>
</tr>
<tr>
<td></td>
<td>0: No cipher suite is being used.</td>
</tr>
<tr>
<td></td>
<td>sslCs_RSA_RC4_56_SHA1: Secure Hash Algorithm-1, 56-bit</td>
</tr>
<tr>
<td></td>
<td>sslCs_RSA_RC4_128_SHA1: Secure Hash Algorithm-1, 128-bit.</td>
</tr>
<tr>
<td></td>
<td>sslCs_RSA_RC4_40_MD5: Rivest Message Digest 5, 40-bit.</td>
</tr>
<tr>
<td></td>
<td>sslCs_RSA_RC4_128_MD5: Rivest Message Digest 5, 128-bit.</td>
</tr>
<tr>
<td>cipher</td>
<td>A constant that represents the cipher that’s being used for this connection:</td>
</tr>
<tr>
<td></td>
<td>sslCsiCipherNull: No cipher currently set.</td>
</tr>
<tr>
<td></td>
<td>sslCsiCipherRc4: RSA RC4.</td>
</tr>
<tr>
<td>digest</td>
<td>A constant that represents the message digest format.:</td>
</tr>
<tr>
<td></td>
<td>sslCsiDigestNull: No message digest.</td>
</tr>
<tr>
<td></td>
<td>sslCsiDigestMd2: Rivest Message Digest 2.</td>
</tr>
<tr>
<td></td>
<td>sslCsiDigestMd5: Rivest Message Digest 5.</td>
</tr>
<tr>
<td></td>
<td>sslCsiDigestSha1: Secure Hash Algorithm-1.</td>
</tr>
<tr>
<td>keyExchange</td>
<td>A constant that represents the key exchange type that’s being used:</td>
</tr>
<tr>
<td></td>
<td>sslCsiKeyExchNull: No key exchange.</td>
</tr>
<tr>
<td></td>
<td>sslCsiKeyExchRsa: RSA.</td>
</tr>
</tbody>
</table>

2168  Palm OS Programmer’s API Reference
### SSL Structures and Data Types

#### SSL Structures

- **authentication**: A constant that represents the authentication type that’s being used:
  - `sslCsiAuthNull`: No key exchange.
  - `sslCsiAuthRsa`: RSA.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>The SSL version number.</td>
</tr>
<tr>
<td><code>cipherBitLength</code></td>
<td>The length of the material used for encryption key generation, in bits. For export ciphers this will be either 40 or 56 bits.</td>
</tr>
<tr>
<td><code>cipherKeyLength</code></td>
<td>The length of the encryption key that’s generated, in bits. For an export RC4 cipher, the <code>cipherKeyLength</code> is 128.</td>
</tr>
<tr>
<td><code>keyExchangeLength</code></td>
<td>The length of the public key used to establish a shared secret, in bits.</td>
</tr>
<tr>
<td><code>authenticationLength</code></td>
<td>The length of the public key used to ensure that the key exchange wasn’t tampered with, in bits. For export ciphers, the <code>keyExchangeLength</code> is often shorter than the <code>authenticationLength</code>.</td>
</tr>
<tr>
<td><code>export</code></td>
<td>A boolean value: 1 if an export cipher is being used; 0 otherwise.</td>
</tr>
</tbody>
</table>

The list of possible cipher suites can be retrieved from the `CipherSuites` attribute. The suite that’s currently being used is given by the `CipherSuite` attribute.

---

**New**

**SslExtendedItem**

Structure that’s used to describe a single certificate-related item. Every `SslExtendedItem` has a distinct (pre-defined) “extended item type.” The location of the item’s data is indicated by the structure—the data isn’t stored in the structure.

You never create `SslExtendedItem` structures yourself. The objects that relate to a particular certificate are collected into an
SslExtendedItems structure and returned to you (principally) by your verify callback function.

```c
typedef struct SslExtendedItem_st {
    UInt16 type;
    UInt16 field;
    UInt16 dataType;
    UInt16 len;
    UInt32 offset;
} SslExtendedItem;
```

The fields are:

- **type**: The extended item type, one of:
  - `sslExItemTypeX509`: X.509 Certificate
  - `sslExItemTypeRSA`: RSA public key
  - `sslExItemTypeRDN`: An X.509 Relative Distinguished Name (RDN). This is the certificate’s name. Each certificate contains two names, the `Subject` of the certificate and the `Issuer` of the certificate. Both are encoded as RDNs that contain multiple fields.
  - `sslExItemTypeX509Ex`: X.509 certificates can contain multiple “extensions.” This type is used to specify that the item is a certificate extension.

- **field**: Type-specific value.

- **dataType**: The encoding for the item’s data. For the SSL library, the value is one of the ASN.1 encoding types, as listed in SslLibAsn1.h. The encoding is relevant if you’re trying to display the data bytes.
New

SslExtendedItems

Structure that contains a set of related data items that pertain to an SSL certificate or other cryptographic entity. The SslExtendedItems structure is used as part of the SslVerify structure, and is used as the type of the PeerCert attribute.

The SslExtendedItems structure is used by the SSL library to return information to your application; you never create and populate an SslExtendedItems structure yourself.

typedef struct SslExtendedItems_st
{
    UInt32  length;
    UInt32  num;
    SslExtendedItem  eitem[1];
} SslExtendedItems;

The fields are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>Total length of the structure, in bytes.</td>
</tr>
<tr>
<td>num</td>
<td>Number of elements in the eitem array.</td>
</tr>
<tr>
<td>eitem</td>
<td>Individual data items (these are the “related data items” mentioned above). The array contains num elements.</td>
</tr>
</tbody>
</table>

The items in the eitem array needn’t all be the same “extended item type” (as defined in SslExtendedItem). For example, the SslExtendedItems structure for a certificate typically contains an sslExItemTypeX509 item, an sslExItemTypeRSA, and an sslExItemType (for the subject name).
SSL Structures

SSL Structures and Data Types

New

SslIoBuf

Structure that represents a data I/O operation. It’s passed to your info callback function as data is being read or written.

```c
typedef struct {
    SslContext  *ssl;
    UInt8      *ptr;
    UInt32     outNum;
    UInt32     inNum;
    UInt32     max;
    UInt32     err;
    UInt32     flags;
} SslIoBuf;
```

You can ask for the info callback function to be invoked just before data is read, just after it’s read, just before it’s written, and just after it’s written. You set the `InfoInterest` attribute to register for these four events.

Most of the SslIoBuf fields depend on which of these four events is being described:

**Fields**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx</td>
<td>The SSL context that performed the operation.</td>
</tr>
<tr>
<td>ptr</td>
<td>A pointer to the data that was just read, or that’s about to be written. For “before read” and “after write” events, the buffer should be empty.</td>
</tr>
<tr>
<td>outNum</td>
<td>The number of bytes that were just read or written. For the “before” events, this field is set to 0.</td>
</tr>
<tr>
<td>inNum</td>
<td>The number of bytes that are about to be read or written. For the “before read” case, this is the minimum number of bytes that the context wants to read—the actual number of bytes read may be larger.</td>
</tr>
</tbody>
</table>
The information in this structure is provided for debugging and informational purposes only. You could use it, for example, to display the progress of an I/O operation.

**New**

**SslLibCallback**

**Purpose** Structure that represents a secure socket.

**Declared In** SsslLib.h

**Prototype**

typedef struct {
    void  *reserved;
    SsslLibCallbackFunc  callback;
    void  *data;
    SsslContext  *ssl;
}  SsslLibCallback

**Fields**

- **callback**
  A pointer to a callback function.

- **data**
  A pointer to data that’s passed to the function when it’s invoked.

- **ssl**
  The SSL context to which the callback applies, or **NULL** if the structure is owned by an SsslLib object.

- **reserved**
  Reserved for future use.
SSL Structures and Data Types

Comments
To register a callback function, create a SslLibCallback structure, fill out the callback and data fields and pass the structure to one of the SslCallbackFunc functions. The structure is copied into the context, and the ssl field is properly set (in the copy).

It’s the caller’s responsibility to free the original SslLibCallback structure. The same SslLibCallback structure can be used to set more than one callback function.

For more information, see SslCallbackFunc in Chapter 80, “SSL Functions.”

New

SslSession
Structure that represents the current SSL session.

typedef struct SslSession_st
{
    UInt32    length;
    UInt16    version;
    unsigned char cipherSuite[2];
    unsigned char compression;
    unsigned char sessionId[33];
    unsigned char masterSecret[48];
    unsigned char time[8];
    unsigned char timeout[4];
    UInt16    certificateOffset;
    UInt16    extraData;
} SslSession;

A context’s SslSession is populated when you call SslOpen. You can retrieve a context’s SslSession structure and modify it directly—particularly if you want to set the fields that are reserved for application use. To retrieve the SslSession, call the SslContextGet_SslSession macro. Keep in mind that this gives you a pointer to the context’s internal SslSession attribute. If you want to store the SslSession, you must copy the attribute’s data.

The fields are:
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>The total size of the structure, in bytes.</td>
</tr>
<tr>
<td>version</td>
<td>Version number of the SSL protocol that’s being used; 0 if the session is using cleartext.</td>
</tr>
<tr>
<td>cipherSuite</td>
<td>Cipher suite that’s being used. See the CipherSuite attribute.</td>
</tr>
<tr>
<td>compression</td>
<td>The name of the compression scheme that’s being used.</td>
</tr>
<tr>
<td>sessionId</td>
<td>Session identification number. The ID value can be as many as 32 bytes long; the first byte gives the number of valid bytes.</td>
</tr>
<tr>
<td>masterSecret</td>
<td>The value of the “master secret” that was established during the SSL handshake.</td>
</tr>
<tr>
<td>time</td>
<td>Used to record the time that the session started, using a local time representation. The format can be anything that fits in eight bytes. Unused by the SSL library, an application can use this field as it wishes.</td>
</tr>
<tr>
<td>timeout</td>
<td>The number of seconds that the session should remain valid. The timeout field is also unused by the SSL library. Use it as you wish.</td>
</tr>
<tr>
<td>certificateOffset</td>
<td>Offset, in bytes, from the start of the structure to an SslExtendedItems structure that contains the server’s certificate. The field is provided for the application’s convenience. The SSL library doesn’t actually copy the certificate: If you want to cache the server’s certificate, you have to retrieve the data through the PeerCert attribute, extend the size of the SslSession structure, copy the certificate data, and then set the certificateOffset yourself.</td>
</tr>
<tr>
<td>extraData</td>
<td>Offset, in bytes, from the start of the structure to some extra data that the application keeps for itself. This is similar to the certificateOffset field: You have to allocate, copy, and mark the data yourself.</td>
</tr>
</tbody>
</table>
New

SslSocket

The SslSocket structure contains information about the socket that’s being used in the network connection. The structure is defined as:

```c
typedef struct {
    NetSocketRef   socket;
    Int16   flags;
    UInt16  addrLen;
    Err    err;
    Int32   timeout;
    NetSocketAddrType   addr;
}  SslSocket;
```

You can create your own SslSocket structure and add its data to a context through the `SsslContextSet_IoStruct` macro. When you do this, the data in your structure is copied into a structure that’s held by the context. Note, however, that the socket field isn’t copied. To set the socket, use `SsslContextSet_Socket`.

The argument values that you pass to `SsslSend` and `SsslReceive` are used to modify the context’s SslSocket structure. If, for example, you create an SslSocket with a particular timeout value, set it in a context, and then call `SsslSend` with a different timeout value (passed as an argument), the latter timeout is recorded in the context’s SslSocket structure, overriding your original setting.

To retrieve a pointer to the context’s SslSocket, call `SsslContextGet_IoStruct`.
The fields are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket</td>
<td>Socket identifier, as created and returned by NetLibSocketOpen. The SSL library doesn’t open or connect the socket for you; all socket operations other than reading and writing data must be done through net library calls. You never set the <code>socket</code> field directly; use the <code>SslContextSet_Socket</code> macro, instead.</td>
</tr>
<tr>
<td>flags</td>
<td>Options that control a socket’s I/O operations. See <code>I/O Flags</code> in Chapter 61, “Net Library,” on page 1413 for a description of these options.</td>
</tr>
<tr>
<td>addr</td>
<td>The most recent address that the socket read from or wrote to.</td>
</tr>
<tr>
<td>addrLen</td>
<td>The length, in bytes, of the <code>addr</code> field.</td>
</tr>
<tr>
<td>err</td>
<td>The most recent socket error.</td>
</tr>
<tr>
<td>timeout</td>
<td>The socket’s I/O operation timeout value, in system ticks.</td>
</tr>
</tbody>
</table>

**New**

**SslVerify**

Structure that’s passed to a verify callback function to provide information about a certificate that’s being verified. This happens when the verification process hits a snag, giving your app a chance to handle the problem. The callback is also invoked when the process has successfully completed.
typedef struct SslVerify_st
{
    SslExtendedItems  *certificate;
    SslExtendedItems  *fieldItems;
    UInt32  field;
    SslExtendedItems  *ex;
    UInt32  depth;
    UInt32  state;
} SslVerify;

The fields are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>certificate</td>
<td>Points to the certificate that’s being verified.</td>
</tr>
<tr>
<td>fieldItems</td>
<td>fieldItems is an array of extended items; field is an index into fieldItems. The SslExtendedItems that’s located at fieldItems[field] contains the set of items that’s causing the problem. (See below.)</td>
</tr>
<tr>
<td>field</td>
<td></td>
</tr>
<tr>
<td>ex</td>
<td>If this is an extension error, the ex field contains the data element of the X509 extension that failed.</td>
</tr>
<tr>
<td>depth</td>
<td>This is the level of the certificate being processed, where 0 is the server’s certificate, and higher numbers are certificates along the “certificate chain” that leads to a trusted root certificate.</td>
</tr>
</tbody>
</table>

The list of SslExtendedItem elements that the SslExtendedItems object located at fieldItems[field] contains depends on the error flavor:
The `asn1ExItemTypeX509Ex` item needs to be decoded in order to be used. The item contains the “object identifier” part of a certificate extension. The item in the next field (i.e. `fieldItems[field+1]`) depends on the item’s `dataType` field: If it’s `asn1Boolean`, than the “+1” item contains the extension’s optional `Boolean` “this is a critical extension” field, otherwise it contains the extension data itself. If “+1” contains the `Boolean`, than “+2” is the data.

The SSL library attempts to interpret only critical extensions, so the critical field should always be present. If a critical extension isn’t understood, the certificate should be rejected.
SSL Attributes and Macros

As described in Chapter 81, “SSL Structures and Data Types,” the SslLib and SslContext objects list and describe the properties, or attributes, of an SSL context.

There are a number of SSL attributes: Some provide general-purpose information that any SSL session needs, some are used only for specific protocols, some are provided for debugging purposes, and so on.

Associated with each attribute is a set of macros that get and set the attribute’s value.

This chapter lists and describes the SSL attributes, how they’re used, the values that they take (including defaults), and the macros that access them:

- “SSL Macro Names” explains how the macro names are formed, and how they correspond to the SSL attributes.
- “SSL Attribute Data Types” talks about the types of data an attribute can represent.
- “SSL Macro Pseudo-Protocol” provides a pseudo-protocol for the macros.
- “SSL Attributes” lists the attributes in alphabetical order.
- “SSL Attribute Constants” is a list of constants that represent the attributes in the attribute-setting functions (which you can use instead of the macros, although it isn’t recommended; the functions are explained below).

SSL Macro Names

An SSL attribute is accessed through a set of (as many as) four macros: There’s a macro that lets you set an attribute’s value in an
SSL Attributes and Macros

SSL Attribute Data Types

SSLLib, another that sets the attribute’s value in an SslContext, and two others that retrieve attribute values from these objects.

The names of the four macros for a given attribute follow this pattern:

- SsslLibSet_Attribute sets the attribute in an SsslLib.
- SsslLibGet_Attribute gets the attribute from an SsslLib.
- SsslContextSet_Attribute sets the attribute in an SsslContext.
- SsslContextGet_Attribute gets the attribute from an SsslContext.

For example, the four macros that set and get the Mode attribute are:

- SsslLibSet_Mode
- SsslLibGet_Mode
- SsslContextSet_Mode
- SsslContextGet_Mode

Note that not all attributes require all four macros: Some attributes are read-only, while others apply only to SsslContext objects.

The macros are covers for the eight attribute-setting functions described in Chapter 80, “SSL Functions,” on page 2135. It’s recommended that you use the macros rather than the functions, but you should be aware of the functions’ existence.

SSL Attribute Data Types

An SSL attribute is either an integer or a pointer. For pointer attributes, the type of the pointed-to data depends on the attribute. The descriptions below explicitly state these data types. (The macro declarations in the SsslLibMac.h file don’t include the data types.)

When setting a pointer attribute, the pointed-to data is (almost always) copied into the SsslLib or SsslContext. The original data needn’t be preserved—the caller is free to destroy the original data when the pointer-setting macro returns. The exceptions to this rule are pointed out in the macro descriptions, below.
To retrieve a pointer attribute, you pass in the address of a pointer; the macro will reset the pointer to data that belongs to the `SslLib` or `SslContext`. You mustn’t free or otherwise modify this data.

The macros that set and get integer attributes are defined in terms of the `SslLibSetLong`, `SslLibGetLong`, `SslContextSetLong`, and `SslContextGetLong` functions, described in Chapter 80, “SSL Functions,” on page 2135. The pointer-accessing macros are covers for an analogous set of functions (`SslLibSetPtr`, et al.). You should rarely need to call these functions directly—they’re meant to be called by the attribute macros.

**SSL Macro Pseudo-Protocol**

This section provides pseudo-protocol for the attribute macros.

---

**New** `SslContextGet_Attribute (integer version)`

**Purpose** Returns the value of an integer-valued attribute that’s stored in an `SslContext` object. This is a cover for the `SslContextGetLong` function.

**Prototype**

```
Int32 SslContextSet_Attribute ( UInt16 libRef, SslContext *context )
```

**Parameters**

- `libRef` (68k only) SSL library reference number.
- `context` The `SslContext` that owns the attribute.

**Result** Returns the attribute’s value, or a negative-valued error code as described in `SslContextGetLong`. 
**SSL Attributes and Macros**

*SSL Macro Pseudo-Protocol*

---

### New

**SslContextGet_Attribute** (pointer version)

**Purpose**

Gets a pointer-valued attribute from an `SslContext` object. This is a cover for the `SslContextGetPtr` function.

**Prototype**

```
Int32 SslContextGet_Attribute ( UInt16 libRef, SslContext *context, void **data )
```

**Parameters**

- `libRef` (68k only) SSL library reference number.
- `context` The `SslContext` that owns the attribute.
- `data` The macro resets `*data` so it points to the attribute’s data. Unless otherwise noted, the pointed-to data belongs to the `SslContext` and shouldn’t be freed.

**Result**

Returns 0 upon success; see `SslContextGetPtr` for error return values.

### New

**SslLibSet_Attribute** (integer version)

**Purpose**

Sets an integer-valued attribute in an `SslContext` object. This is a cover for the `SslLibSetLong` function.

**Prototype**

```
Err SslLibSet_Attribute ( UInt16 libRef, SslContext *context, Int32 value )
```

**Parameters**

- `libRef` (68k only) SSL library reference number.
- `context` The `SslContext` that owns the attribute.
- `value` The value you want to set the attribute to.

**Result**

Returns 0 upon success; see `SslLibSetLong` for error return values.
SSL Attributes and Macros
SSL Macro Pseudo-Protocol

New

SslContextSet_Attribute (pointer version)

Purpose
Sets a pointer-valued attribute in an SslContext object. This is a cover for the SslContextSetPtr function.

Prototype
Err SslContextSet_Attribute ( UInt16 libRef,
SslContext *context, void *data )

Parameters
- libRef (68k only) SSL library reference number.
- context The SslContext that owns the attribute.
- data A pointer to the data that you want to set.

Result
Returns 0 upon success; see SslContextSetPtr for error return values.

Comments
The data from data is copied into the SslContext. You can free the original data after the macro returns.

New

SslLibGet_Attribute (integer version)

Purpose
Returns the value of an integer-valued attribute that’s stored in an SslLib object. This is a cover for the SslLibGetLong function.

Prototype
Int32 SslLibSet_Attribute ( UInt16 libRef,
SslLib *lib )

Parameters
- libRef (68k only) SSL library reference number.
- lib The SslLib that owns the attribute.

Result
Returns the attribute’s value, or a negative-valued error code.
SSL Attributes and Macros
SSL Macro Pseudo-Protocol

New

**SslLibGet_Attribute (pointer version)**

**Purpose**
Gets a pointer-valued attribute from an SslLib object. This is a cover for the SslLibGetPtr function.

**Prototype**
Int32 SslLibGet_Attribute ( UInt16 libRef, SslLib *lib, void **data )

**Parameters**
- *libRef* (68k only) SSL library reference number.
- *lib* The SslLib that owns the attribute.
- *data* The macro resets *data so it points to the attribute’s data. Unless otherwise noted, the pointed-to data belongs to the SslLib and shouldn’t be freed.

**Result**
Returns 0 upon success; see SslLibGetPtr for error return values.

New

**SslLibSet_Attribute (integer version)**

**Purpose**
Sets an integer-valued attribute in an SslLib object. This is a cover for the SslLibSetLong function.

**Prototype**
Err SslLibSet_Attribute ( UInt16 libRef, SslLib *lib, Int32 value )

**Parameters**
- *libRef* (68k only) SSL library reference number.
- *lib* The SslLib that owns the attribute.
- *value* The value you want to set the attribute to.

**Result**
Returns 0 upon success; see SslLibSetLong for error return values.
SSL Attributes and Macros

SSL Attributes

New

SslLibSet_Attribute (pointer version)

Purpose
Sets a pointer-valued attribute in an SslLib object. This is a cover for the SslLibSetPtr function.

Prototype
Err SslLibSet_Attribute ( UInt16 libRef, SslLib *lib, void *data )

Parameters
- `libRef` (68k only) SSL library reference number.
- `lib` The SslLib that owns the attribute.
- `data` A pointer to the data that you want to set.

Result
Returns 0 upon success; see SslLibSetPtr for error return values.

Comments
The data from data is copied into the SslLib. You can free the original data after the macro returns.

SSL Attributes

New

AppInt32

Convenient Int32 datum that the application can use for whatever purpose it wants—as an object ID, for example. Note that the value is not copied from an SslLib into the SslContexts that it spawns.

type: Int32

macros:
- SslLibSet_AppInt32
- SslLibGet_AppInt32
- SslContextSet_AppInt32
- SslContextGet_AppInt32
New 

**AppPtr**

Convenient pointer that the application can use for whatever purpose it wants.

type:  (void *)

The application retains ownership of the pointed-to data. The data is *not* copied into the `SslLib` or `SslContext`.

macros:  `SslLibSet_AppPtr`
          `SslLibGet_AppPtr`
          `SslContextSet_AppPtr`
          `SslContextGet_AppPtr`

---

New 

**AutoFlush**

Enables/disables automatic flushing of data after an `SslSend` or `SslWrite`.

type:  `Int32`

values:  0:  Autoflush is turned off.
          1:  Autoflush is turned on.

default:  1 (autoflush on)

macros:  `SslLibSet_AutoFlush`
          `SslLibGet_AutoFlush`
          `SslContextSet_AutoFlush`
          `SslContextGet_AutoFlush`

When enabled, `SslFlush` is automatically called after every call to `SslSend` and `SslWrite`. If you have autoflush disabled, you have to call the `SslFlush` function yourself. You don’t have to call it after *every* `SslSend` or `SslWrite`, but you *do* have to call it before the `SslContext`’s write buffer fills up.

The write buffer’s size is given by `WbufSize`; the amount of data that’s currently in the buffer is given by `WriteBufPending`
New

BufferedReuse

Allows the last message in an SSL handshake to be buffered. This only applies if SessionReused is true (non-zero).

type: Int32

values: 0: Don’t buffer the last handshake message.
non-zero: Buffer the last handshake message.

macros: SslContextGet_BufferedReuse

As described in SessionReused, the handshake involved in re-establishing a previous session can be a truncated version of a normal handshake. At the end of the truncated handshake, the SSL library sends a message to the server. The BufferReuse attribute, if enabled, will buffer the last message of the reused handshake instead of sending it over the network. The message will be sent when the application sends its first “real” data.

If you have BufferReuse enabled, you must make sure that the final handshake message actually gets sent before reading any incoming data.

There are security implications, here, in that a man-in-the-middle attack could only be detected once the first data bytes are read from the server. In other words, an attacker could have read all the bytes in the first “real data” message sent to the server. For this reason this attribute should not normally be used.
**SSL Attributes and Macros**

**SSL Attributes**

### CipherSuite

New

Represents the cipher suite that’s currently in use in this context. _SslContext_ only, read only.

**type:** (UInt8 *)

**values:** Each suite is represented by a two-byte value.

- 0: No cipher suite is being used.
- `sslCs_RSA_RC4_56_SHA1`: Secure Hash Algorithm-1, 56-bit
- `sslCs_RSA_RC4_128_SHA1`: Secure Hash Algorithm-1, 128-bit.
- `sslCs_RSA_RC4_40_MD5`: Rivest Message Digest 5, 40-bit.
- `sslCs_RSA_RC4_128_MD5`: Rivest Message Digest 5, 128-bit.

**macros:** `SslContextGet_CipherSuite`

**See Also**  
[CipherSuites](#)

### CipherSuiteInfo

New

Returns information about the cipher suite or certificate that’s currently being used. _SslContext_ only, read only.

**type:** *(SslCipherSuiteInfo *)

The `SslCipherSuiteInfo` structure must be allocated before it’s passed in. The caller retains ownership of the structure.

**values:** The structure is populated by the context. For information on what the structure contains, see `SslCipherSuiteInfo`.

**macros:** `SslContextGet_CipherSuiteInfo`
To return *just* the cipher suite (without the surrounding information), get the `CipherSuite` attribute.

### New

**CipherSuites**

Contains a list of cipher suites that the client supports, in order of preference.

**type:** (UInt8 *)

**values:** Each element in the list takes two bytes. The first two byte element give the sizes of the rest of the list, in bytes (i.e. the “size bytes” themselves aren’t counted in the measurement). The rest of the list is made up of the following (two-byte) constants:

- `sslCs_RSA_RC4_128_MD5`
- `sslCs_RSA_RC4_128_SHA1`
- `sslCs_RSA_RC4_56_SHA1`
- `sslCs_RSA_RC4_40_MD5`

The data in the list is given in network byte order.

**default:** The default list contains {0x00, 0x08} followed by the suites listed above, in order. To restore the default list, pass a NULL pointer.

**macros:**

- `SslLibSet_CipherSuites`
- `SslLibGet_CipherSuites`
- `SslContextSet_CipherSuites`
- `SslContextGet_CipherSuites`

As an example, the following code creates a list of the strong encryption suites (only) and sets it into an `SslLib`:

```c
static UInt8 cipherSuites[] = {
    0x00, 0x04,
    sslCs_RSA_RC4_128_MD5,
    sslCs_RSA_RC4_128_SHA1
};

SslLibSet_CipherSuites(lib, cipherSuites);
```
SSL Attributes and Macros
SSL Attributes

The cipher suites list is used when the client and server negotiate for an acceptable suite. The suite that’s chosen is represented by the CipherSuite attribute.

New

Compat

Allows compatibility with malformed or incorrect SSL messages sent from the server

type: Int32

values: 0: No incompatibility allowed.

If non-zero, the value should be a combination of the following flags.

sslCompatReuseCipherBug: Allows servers to change cipher suites on session-reuse.

sslCompatNetscapeCaDnBug: Supports old versions of Netscape servers that encode “Distinguished Names” certificates incorrectly. (Note that this isn’t currently an issue since the SSL library doesn’t support client certificates.)

sslCompat1RecordPerMessage: Ensures that SSL messages aren’t broken across records. (Some servers don’t like to receive SSL messages separated into multiple SSL records.)

sslCompatBigRecords: Supports records larger than 16k bytes. (In SSLv3, record size is normally limited to 16k.)

sslCompatAll: Enables all the foregoing.

default: 0

macros: SslLibSet_Compat
SslLibGet_Compat
SslContextSet_Compat
SslContextGet_Compat
**New**

**DontSendShutdown**

Avoid sending a shutdown message.

**type:** Int32

**values:**

- 0:   Send the shutdown message.
- non-zero:   Don’t send the shutdown message.

**default:** 0

**macros:**

- SslLibSet_DontSendShutdown
- SslLibGet_DontSendShutdown
- SslContextSet_DontSendShutdown
- SslContextGet_DontSendShutdown

When `SslClose` is called, the two SSL endpoints swap shutdown messages. You can suppress transmission of the local message by setting this attribute.

To ignore the remote shutdown message, set the `DontWaitForShutdown` attribute.

**New**

**DontWaitForShutdown**

Avoid waiting for reception of the server’s shutdown message.

**type:** Int32

**values:**

- 0:   Wait for the shutdown message.
- non-zero:   Don’t wait the shutdown message.

**default:** 0

**macros:**

- SslLibSet_DontWaitForShutdown
- SslLibGet_DontWaitForShutdown
- SslContextSet_DontWaitForShutdown
- SslContextGet_DontWaitForShutdown

See `DontSendShutdown` for details.
### New Error

Stores the most recent fatal error code. `SslContext` only.

**type:** `Int32`  
**values:**  
- `0`: No error  
- non-zero: An error occurred somewhere up to here. Note that SslErrIo errors are exempt (they’re not considered to be fatal).

**default:** `0`  
**macros:**  
- `SslContextSet_Error`  
- `SslContextGet_Error`

After you read an `Error` value, you should set the attribute’s value back to `0`. The value is reset during an SSL Reset, but it otherwise holds the latest fatal error code.

### New HsState

Current SSL state. `SslContext` only, read only. Used primarily for debugging.

**type:** `Int32`  
**values:** Defined by the SSL protocol.  
**macros:**  
- `SslContextGet_HsState`

### New InfoCallback

Identifies the callback function that’s called at particular moments in an SSL transaction.

**type:** `(SslCallback *)`
If you set the `InfoCallback` attribute, you must also set the `InfoInterest` attribute to register for the notifications that you want.

See the `SslCallbackFunc` description in Chapter 80, “SSL Functions,” on page 2135 for more information about the callback functions.
SSL Attributes and Macros

SSL Attributes

New

**InfoInterest**

Specifies the events for which the callback function identified by the *InfoCallback* attribute will be called.

**type:** Int32

**values:** The value is formed as a combination of the following (or 0 if you don’t want your callback to be called):

- `sslFlgInfoAlert`: The callback wants to know about alert notifications. In an alert, the function receives `sslArgInfoAlert` as its command argument.

- `sslFlgInfoHandshake`: The callback wants to know about handshake notifications (`sslArgInfoHandshake`).

- `sslFlgInfoIO`: The callback wants to be called before and after each *SslCallback* operation (`sslArgInfoReadBefore, sslArgInfoReadAfter, sslArgInfoWriteBefore, sslArgInfoWriteAfter`).

- `sslFlgInfoCert`: The callback wants to know about certificate notifications (`sslArgInfoCert`).

**default:** 0

**macros:** `SslLibSet_InfoInterest`  
`SslLibGet_InfoInterest`  
`SslContextSet_InfoInterest`  
`SslContextGet_InfoInterest`

For these to be effective, you must set an info callback function (see *InfoCallback*).

For more information on how the callback functions work, and for detailed specifications of the command values referred to above, see the *SslCallbackFunc* description in Chapter 80, “SSL Functions,” on page 2135.
**New**

**IoTimeout**

Amount of time that an I/O operation should wait for a response from the other side of the network connection before it gives up and returns an error. The timeout is measured in ticks.

*type:* Int32  
*values:* [0, 0xefffffff].  
*default:* 10 seconds  
*macros:* SslContextSet_IoTimeout  
SslContextGet_IoTimeout

Individual SSL library functions can specify their own timeout values. In the absence of a timeout specification, the IoTimeout value is used.

**New**

**IoFlags**

Socket I/O flags that are used by the data read and write functions. *SslContext* only.

*type:* Int32  
*values:* See the “I/O Flags” on page 1421 for a list of these flags.  
netIOFlagOutOfBand and netIOFlagPeek don’t apply; they’re twiddled out of the value you supply.  
*default:* none  
*macros:* SslContextSet_IoFlags  
SslContextGet_IoFlags
New

**IoStruct**

_SslSocket_ structure that holds arguments that are passed to the underlying net library functions. _SslContext_ only.

**type:** \((SslSocket \ *)\)

**values:** See _SslSocket_.

**default:** \((SslSocket \ *)\) NULL

**macros:** SslContextSet_IoStruct
               SslContextGet_IoStruct

When you call SslContextSet_IoStruct, the socket field of the _SslSocket_ that you pass in is ignored. Use the _Socket_ attribute to set the socket.

The structure’s other fields correspond to the arguments to the _SslSend_ and _SslReceive_ functions (which see for details). When you call one of these functions, the arguments that you pass are copied into the context’s _SslSocket_ structure. Thus, the _SslSocket_ configuration that you set through SslContextSet_IoStruct may be updated when you call an I/O function.

New

**LastAlert**

Most recent fatal error code. _SslContext_ only, read only.

**type:** Int32

**values:** See the sslAlertType constants in SslLibMac.h.

**default:** 0

**macros:** SslContextGet_LastAlert

An alert is an error or status code that’s sent by the server. Alerts can be fatal or non-fatal. Fatal alerts are in the form 0x02xx; non-fatal alerts are 0x01xx, as shown below:
```c
Int32 alert;

SslContextGet_LastAlert( context, &alert);

if ( alert != 0 ) {
    if ( alert & 0x0f00 == 0x0200 )
        /* fatal */
    else if ( alert & 0x0f00 == 0x0100 )
        /* non-fatal */
    else
        /* undefined */
}
```

**New**

**LastApi**

Represents the most recently called SSL library function. 
**SslContext** only, read only.

**type:** `Int32`

**values:**

- `sslLastApiNone`: No SSL library function call in this context since the last SSL reset.
- `sslLastApiRead`: The previous function was **SslOpen**.
- `sslLastApiRead`: The previous function was **SslRead, SslSeek**, or **SslReceive**.
- `sslLastApiWrite`: The previous function was **SslWrite** or **SslSend**.
- `sslLastApiFlush`: The previous function was **SslFlush**.
- `sslLastApiShutdown`: The previous function was **SslClose**.

**macros:** `SslContextGet_LastApi`
**New**  
**LastIO**  
Represents the nature of the previous I/O operation. `SslContext` only, read only.  
**type:** Int32  
**values:**  
- `sslLastIoNone`: There is yet to be an I/O operation in this context since the last SSL reset.  
- `sslLastIoRead`: The previous I/O operation was a read.  
- `sslLastIoWrite`: The previous I/O operation was a write.  
**default:** `sslLastIoNone`  
**macros:** `SslContextGet_LastIO`  
This attribute can be useful when you’re determining the reason for an error.

**New**  
**Mode**  
Turns SSL on and off.  
**type:** Int32  
**values:**  
- `sslModeClear`: SSL is turned off.  
- `sslModeSsl`: SSL is turned on  
- `sslModeSslClient`: SSL is turned on, and this endpoint is a client.  
- `sslModeFlush`: Data-clearing flag. If present, the context’s internal data buffers are cleared (the data is lost). This flag is applicable to `SslContext` only.
default: sslModeSslClient

macros: SslLibSet_Mode
        SslLibGet_Mode
        SslContextSet_Mode
        SslContextGet_Mode

sslModeSsl and sslModeSslClient are essentially the same; the only difference is that the latter explicitly declares the object to represent the client side of the connection. In Palm OS 5, SSL endpoints are always clients.

When set to sslModeClear, SSL is bypassed. This lets you write your application with the SSL library API and still perform normal (non-SSL) data transfers. Your application will still get the advantage of the data buffering provided by the SSL I/O functions (SslRead, SslWrite, et al.).

You can use sslModeSsl as a mask to determine if SSL is on:

If (SslContextGet_Mode(ssl) & sslModeSsl)
    /* SSL protocol enabled */
else
    /* Using cleartext */

The Mode Attribute in SslContexts

Setting an SslContext’s Mode (even if you set it to the status quo) causes an “SSL reset.” This sets most of the other SSL attributes back to their default values, and puts the SslContext in the proper state to open a new SSL session. Note that an SSL reset doesn’t clear SSL session information, thus allowing the previous session to be used in a new session with the same server. See ReadStreaming for details.

To force an SSL reset, do this:

SslContextSet_Mode(context,SslContextGet_Mode(context));

Switching from SSL-off to SSL-on initiates a new SSL handshake.

sslModeFlush is a flag that, when present, clears the internal read and write buffers. Any data in the buffers is lost. You would normally use this flag when you’re reusing an SSL session.
SSL Attributes and Macros

SSL Attributes

New

PeerCert

Structure that represents the certificate that was supplied by the server. SslContext only, read only.

type: \( (\text{SslExtendedItems} \ast) \)

values: If a certificate is available, the context passes back a pointer to a structure. See SslExtendedItems for details.

default: \( (\text{SslExtendedItems} \ast)\) NULL

macros: SslContextGet_PeerCert

New

PeerCommonName

Structure that contains information that you use to retrieve the server’s common name from a previously retrieved certificate. SslContext only, read only.

type: \( (\text{SslExtendedItem} \ast) \)

values: See the example, below.

default: \( (\text{SslExtendedItem} \ast)\) NULL

macros: SslContextGet_PeerCommonName

To use this attribute, you must also retrieve the server’s certificate through SslContextGet_PeerCert. You then perform some pointer juggling that’s best explained through an example:

```c
SslExtendedItems *cert;
SslExtendedItem *commonName;

/* The length of the common name */
Int16 length;

/* A pointer to the beginning of the common name. */
Int8 *bytes;

SslContextGet_PeerCert( ssl, &cert );
```
if ( cert != NULL ) {
    SslContextGet_PeerCommonName( ssl, &commonName );

    /* Get the common name length from the SslExtendedItem
     * struct.
     */
    length = commonName->len;

    /* The name itself is in the peer certificate, located
     * at an offset (into the SslExtendedItems struct) as
     * indicated by the offset field of the SslExtendedItem
     * struct.
     */
    bytes = ((Int8 *)cert) + commonName->offset;

    /* Now that we have a pointer to the name and the
     * length of the name, we can compare it to the
     * expected value.
     */
    StrNCompare ( bytes, expectedValue, length );
}

If you’re using SSL in an https context, for example, the client
application should ensure that the common name contained in the
server’s certificate matches the requested URL.

▼

**New**

**ProtocolVersion**

The SSL protocol that’s being used

**type:** Int32

**values:** sslVersionSSLv3 only; if you set the protocol to
some other value, SSL won’t work.

**macros:** SslLibSet_ProtocolVersion
             SslLibGet_ProtocolVersion
             SslContextSet_ProtocolVersion
             SslContextGet_ProtocolVersion
SSL Attributes and Macros

SSL Attributes

New

RbufSize

The size of the internal read buffer, in bytes.

type: Int32

values: [0, 16384]

default: 2048

macros: SslLibSet_RbufSize
SslLibGet_RbufSize
SslContextSet_RbufSize
SslContextGet_RbufSize

The actual size of the read buffer may be different from that which you request: The buffer is automatically increased in size if the SSL protocol and/or decryption algorithm demand it. For example, if decryption can only operate on an entire record, and if it’s passed a record that’s longer than the read buffer size, the buffer will grow to accommodate the record.

The ReadStreaming attribute describes an advanced use of the read buffer that decreases data reading latency.

New

ReadBufPending

Returns the number of bytes that are waiting to be read from the context’s read buffer. The measurement includes in-coming SSL encryption data. SslContext only; read only.

type: Int32

values: [0, 16k]

macros: SslContextGet_ReadBufPending

This attribute is provided for debugging purposes.
New

**ReadOutstanding**

Returns the number of bytes that remain to be read from the current SSL record. `SslContext` only; read only.

**type:** Int32

**values:** [0, 16k]

0 means the entire record has been read and verified.

**macros:** SslContextGet_ReadOutstanding

See ReadStreaming for a justification of this attribute.

New

**ReadRecPending**

Returns the number of “real” bytes of data in the context’s read buffer. The measurement doesn’t include SSL encryption data. `SslContext` only; read only.

**type:** Int32

**values:** [0, 16k]

**macros:** SslContextGet_ReadRecPending

If this attribute is 0, the next `SslRead` or `SslReceive` will cause a `NetLibReceive` invocation.

New

**ReadStreaming**

Allows partial record data to be read.

**type:** Int32

**values:**

0: Always wait for an entire record before reading.

non-zero: Don’t wait for an entire record.
SSL Attributes and Macros

SSL Attributes

If the network is running at a very low rate (such as a 300 baud modem), you may want to allow in-coming data to be returned to the application before a full record has been downloaded. Notice, however, that “stream reading” has an implication on security: Since you can’t verify the record until it has been fully read, you must be careful what you do with stream read data. You should never respond to in-coming data until an entire record has been read and verified. The ReadOutstanding attribute can be used to determine if a record has been fully read and verified.

The SSL library may reject stream reading requests. See the Streaming attribute for details.

To retrieve streaming data, use the SslPeek and SslConsume functions (rather than SslRead or SslReceive).

**New**

SessionReused

Indicates whether the context is reusing a previous session. SslContext only, read only.

| type:    | Int32          |
| values:  |               |
| 0:       | No, this is a new session. |
| non-zero:| Yes, this session is a continuation of a previous session. |

| default: | 0 |

| macros:  | SslContextGet_SessionReused |

SSL can re-establish a previously formed session if both ends of the network connection have a common notion of the session’s parameters (as represented by the SslSession attribute). Reusing a session lets the SSL protocol use a truncated (faster) handshake.
SSL Attributes and Macros

SSL Attributes

New

Socket

Specifies the net library socket that an SslContext will use in I/O operations. SslContext only.

- **type:** NetSocketRef (Int16)
- **default:** none
- **macros:** SslContextSet_Socket, SslContextGet_Socket

An SSL context can’t perform any network operations until it’s supplied with a valid, open, connected NetSocketRef. The SSL library doesn’t perform any Net library operations Creation, initialization, and destruction of the socket that the NetSocketRef represents is the application’s responsibility.

This call is used to specify the net library socket that the SslContext should use to perform it’s network I/O operations. A SslContext is unable to perform any network operation until the application creates and supplies a suitable NetSocketRef.

IMPORTANT: The SslLib library uses the NetSocketRef to send data and receive data only—it doesn’t call any other net library functions except for NetLibSend and NetLibReceive. All socket creation, configuration, and shutdown operations must be performed by the application.

New

SslSession

Structure that represents the current or impending SSL session. SslContext only.

- **type:** (SslSession *)
SSL Attributes and Macros

SSL Attributes

values: When a session is opened, the context populates the structure with information. See \texttt{SslSession} for details.

macros: \texttt{SslContextSet\_SslSession}
\texttt{SslContextGet\_SslSession}

You can fine-tune an impending session by creating and populating an \texttt{SslSession} that you pass in through \texttt{SslContextSet\_SslSession}. As with most pointer attributes, the structure is copied into the context. The next time you call \texttt{SslOpen}, the embedded \texttt{SslSession} information is used to configure the session. The structure is then modified, if necessary, to reflect the actual state of the session.

Session information isn’t cleared in an SSL reset. Maintaining the session info allows a session to be reused, as discussed in \texttt{SessionReused}.

\textbf{New} \texttt{SslVerify}

Structure that’s used during certificate verification. \texttt{SslContext} only, read only.

type: \texttt{(SslVerify *)}

values: See the example, below.

default: \texttt{(SslVerify *)NULL}

macros: \texttt{SslContextGet\_SslVerify}

If certification runs into a problem that can’t be resolved by the verify callback function (see \texttt{VerifyCallback}), or if you haven’t registered a verify callback function, an error will be returned to your application, which can retrieve the \texttt{SslVerify} structure in an attempt to solve the conflict itself.

If your application determines that the error isn’t fatal, it should clear the error (through \texttt{SslContextSet\_Error}) and reinvoke the function that returned the error.
New

Streaming

Tells you if the context is stream reading. SslContext only; read only.

type: Int32

default: none

values: 0: The context isn’t stream reading.
non-zero: The context is stream reading.

macros: SslContextGet(Streaming)

In some cases, the SSL library may decide that it doesn’t want to do stream reading, regardless of the value of the ReadStreaming attribute. The value of this attribute tells you the truth.

New

VerifyCallback

Identifies the callback function that’s used during certificate verification.

type: (SslCallback *)

default: none

macros: SslLibSet_VerifyCallback
SslLibGet_VerifyCallback
SslContextSet_VerifyCallback
SslContextGet_VerifyCallback

For more information, see the SslCallbackFunc description in Chapter 80, “SSL Functions.”
**SSL Attributes and Macros**

**SSL Attribute Constants**

---

**New**

**WbufSize**

The size of the internal write buffer, in bytes. Note that SSL protocol overhead takes about 30 bytes, so the “writable” portion of the write buffer will be slightly smaller than the value of `WbufSize`.

- **type:** Int32
- **values:** [0, 16384]
- **default:** 1024
- **macros:**
  - `SslLibSet_WbufSize`
  - `SslLibGet_WbufSize`
  - `SslContextSet_WbufSize`
  - `SslContextGet_WbufSize`

---

**New**

**WriteBufPending**

Returns the number of bytes of data in the context’s write buffer (i.e. waiting to be sent to the remote endpoint). `SsslContext` only; read only.

- **type:** Int32
- **values:** [0, 16k]
- **macros:** `SsslContextGet_WriteBufPending`

This attribute should be zero unless `AutoFlush` is disabled. You must explicitly flush (through `SsslFlush`) before the value of `WriteBufPending` reaches the value of `WbufSize` (the maximum size of the write buffer).

---

**SSL Attribute Constants**

As explained earlier, each of the attribute macros is a cover for one of the eight attribute-setting or -getting functions. (The functions are described in Chapter 80, “SSL Functions,” on page 2135.) Instead of
invoking the attribute macros, you can call the attribute functions, instead.

Unlike the macros, the functions don’t correspond to specific attributes. To tell the function which attribute you want it to access, you pass in one of the attribute constants. These constants usually take the form

\[ \text{sslAttrLibAttribute} \]

or

\[ \text{sslAttrAttribute} \]

where the former represents the attribute in an \textit{SslLib} object, and the latter is the \textit{SslContext} version (note the lack of a qualifying “Context” in the names of the \textit{SslContext} macro constants). As examples:

- \text{sslAttrLibMode} represents the \textit{Mode} attribute in an \textit{SslLib} object.
- \text{SslAttrMode} is the \textit{Mode} attribute in an \textit{SslContext}.
- \text{sslAttrLibAutoFlush} and \text{sslAttrAutoFlush} signify the \textit{AutoFlush} attribute in the \textit{SslLib} and \textit{SslContext}.

Note that there are a few exceptions to this formula.

Alphabetical lists of the attribute constants are given below, one list for \textit{SslLib} attributes, and a second for \textit{SslContext}. The constant names that don’t conform to the formula, above, are noted.

**\textit{SslLib Attribute Constants}**

\[
\begin{align*}
\text{sslAttrLibAppInt} \\
\text{sslAttrLibAppPtr} \\
\text{sslAttrLibAutoFlush} \\
\text{sslAttrLibBufferedReuse} \\
\text{sslAttrLibCompat} \\
\text{sslAttrLibDontSendShutdown} \\
\text{sslAttrLibDontWaitForShutdown} \\
\text{sslAttrLibInfoCallback} \\
\text{sslAttrLibInfoInterest} \\
\text{sslAttrLibMode} \\
\text{sslAttrLibProtocolVersion} \\
\text{sslAttrLibRbufSize} \\
\text{sslAttrLibReadStreaming} \\
\text{sslAttrLibSessionCallback} \\
\text{sslAttrLibVerifyCallback}
\end{align*}
\]
SSL Attributes and Macros

SSL Attribute Constants

<table>
<thead>
<tr>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>sslAttrLibWbufSize</td>
</tr>
</tbody>
</table>

SSLContext Attribute Constants

<table>
<thead>
<tr>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>sslAttrAppInt</td>
</tr>
<tr>
<td>sslAttrAppPtr</td>
</tr>
<tr>
<td>sslAttrAutoFlush</td>
</tr>
<tr>
<td>sslAttrBufferedReuse</td>
</tr>
<tr>
<td>sslAttrCertPeerCert -&gt; PeerCert</td>
</tr>
<tr>
<td>sslAttrCertPeerCommonName -&gt; PeerCommonName</td>
</tr>
<tr>
<td>sslAttrCertSslVerify -&gt; SslVerify</td>
</tr>
<tr>
<td>sslAttrCompat</td>
</tr>
<tr>
<td>sslAttrCspCipherSuiteInfo -&gt; CipherSuiteInfo</td>
</tr>
<tr>
<td>sslAttrCspCipherSuites -&gt; CipherSuites</td>
</tr>
<tr>
<td>sslAttrCspCipherSuites -&gt; CipherSuite</td>
</tr>
<tr>
<td>sslAttrCspSslSession -&gt; SslSession</td>
</tr>
<tr>
<td>sslAttrDontSendShutdown</td>
</tr>
<tr>
<td>sslAttrDontWaitForShutdown</td>
</tr>
<tr>
<td>sslAttrError</td>
</tr>
<tr>
<td>sslAttrHsState</td>
</tr>
<tr>
<td>sslAttrInfoCallback</td>
</tr>
<tr>
<td>sslAttrInfoInterest</td>
</tr>
<tr>
<td>sslAttrIoFlags</td>
</tr>
<tr>
<td>sslAttrIoSocket -&gt; Socket</td>
</tr>
<tr>
<td>sslAttrIoStruct</td>
</tr>
<tr>
<td>sslAttrIoTimeout</td>
</tr>
<tr>
<td>sslAttrLastAlert</td>
</tr>
<tr>
<td>sslAttrLastApi</td>
</tr>
<tr>
<td>sslAttrLastIo</td>
</tr>
<tr>
<td>sslAttrMode</td>
</tr>
<tr>
<td>sslAttrProtocolVersion</td>
</tr>
<tr>
<td>sslAttrRbufSize</td>
</tr>
<tr>
<td>sslAttrReadBufPending</td>
</tr>
<tr>
<td>sslAttrReadOutstanding</td>
</tr>
<tr>
<td>sslAttrReadRecPending</td>
</tr>
<tr>
<td>sslAttrReadStreaming</td>
</tr>
<tr>
<td>sslAttrSessionReused</td>
</tr>
<tr>
<td>sslAttrStreaming</td>
</tr>
<tr>
<td>sslAttrVerifyCallback</td>
</tr>
<tr>
<td>sslAttrWbufSize</td>
</tr>
<tr>
<td>sslAttrWriteBufPending</td>
</tr>
</tbody>
</table>
SSL Error Codes

The SSL functions described in Chapter 80, “SSL Functions,” return errNone if successful, and non-zero otherwise. The table below lists and explains these “non-zero” values.

In addition to the errors listed below, the SSL functions also pass back errors that are returned by the underlying network library (NetLib) functions.

**WARNING!** The sslErrOk error code, declared in SslLib.h is does not indicate success. The error-returning SSL functions return errNone for success.

The errors are grouped under topical headers, and listed alphabetically therein. The groups are:

- **SSL Function Protocol Errors** indicate a malformed function call, or other non-SSL dilemma.
- **SSL Alerts** are errors that your application may need to explore to find out what really happened.
- **SSL Handshake Errors** are returned while SSL handshake messages are being passed between the client and the server.
- **SSL Cryptography Errors** indicate that a cryptographic operation has failed.
- **SSL Illegal Message Errors** are returned when SSL receives an unexpected message type.
- **SSL Certificate Errors** describe the different failures during certificate verification.

**SSL Function Protocol Errors**

sslErrBadArgument A function argument was (generally) invalid.
SSL Error Codes

SSL Alerts

sslErrBadLength
A buffer length argument to a data-reading or -writing function was invalid.

sslErrBadOption
This means the same as sslErrBadArgument.

sslErrBufferTooSmall
A buffer supplied to this function wasn’t big enough for the data that the functions wanted to stuff into it.

sslErrEof
You attempted to read or write a socket that isn’t open.

sslErrFailed
General, unspecified error.

sslErrNullArg
You passed in a NULL argument. This is returned, for example, if you pass in a NULL pointer where a valid SslLib or SslContext is expected.

sslErrOutOfMemory
Not enough memory to allocate the resources that are needed by this function.

SSL Alerts

sslErrCbAbort
Returned by a callback function to indicate that the caller should exit. See SslCallbackFunc in Chapter 80, “SSL Functions.”

sslErrIo
Indicates that an underlying NetLib function has returned a non-fatal error, such as a timeout. When you get this error, you should try to determine what the exact error is—you usually start by looking at the context’s Compat attribute. After you clear the error, you can try to call the function again.

sslErrFatalAlert
An SSL fatal alert was received. To handle the alert, see the LastAlert attribute in Chapter 82, “SSL Attributes and Macros.”
SSL Handshake Errors

sslErrBadPeerFinished
The final check of the SSL handshake failed. This means that there was a problem establishing a shared secret value. It could be caused by the server using a certificate that does not match its private key.

sslErrExtraHandshakeData
An SSL handshake message contained data that shouldn’t have been there.

sslErrHandshakeEncoding
An error occurred while an SSL handshake message (from the server) was being decoded.

sslErrHandshakeProtocol
An error occurred while processing a decoded SSL handshake message.

sslErrReadAppData
A handshake message was expected, but application data was read.

SSL Cryptography Errors

sslErrCsp
General cryptography error.

sslErrDivByZero
Math error. This can happen if a certificate has an invalid public key.

sslErrNoModInverse
Another math error. See above.

sslErrNoRandom
There was a problem with the random number source.
SSL Error Codes

SSL Illegal Message Errors

sslErrBadSignature
An invalid signature was found on an ephemeral Cipher Suite message.

sslErrWrongMessage
An invalid or inappropriate SSL message was received.

sslErrUnexpectedRecord
The wrong type of record was received.

sslErrRecordError
An invalid record was received by the SslContext.

SSL Certificate Errors

sslErrBadDecode
Something went wrong while decoding values during certificate verification.

sslErrCert
General certificate error.

sslErrCertDecodeError
The server’s certificate could not be decoded.

sslErrUnsupportedCertType
The server sent a certificate that isn’t supported (it contains a public key that can’t be decoded).

sslErrUnsupportedSignatureType
The server sent a certificate that has an unrecognized signature type.

sslErrVerifyBadSignature
The certificate’s signature is invalid

sslErrVerifyConstraintViolation
The certificate violates an X509 extension.

sslErrVerifyNotAfter
The certificate has expired.
sslErrVerifyNotBefore
   The certificate is too early (the timestamp window is in the future).

sslErrVerifyNoTrustedRoot
   A trusted certificate store (necessary for certificate verification) couldn’t be found.

sslErrVerifyUnknownCriticalExtension
   An X509 extension (that’s marked as “critical”) isn’t understood by the certificate verification routines.
SMS Exchange Library

This chapter describes the SMS Exchange Library API declared in the header file SmsLib.h. It discusses the following topics:

- SMS Exchange Library Data Structures
- SMS Exchange Library Constants

You interact with the SMS Exchange Library using the Exchange Manager APIs described in Chapter 57, “Exchange Manager,” on page 1297 of this book. For further information on using Exchange Manager, see “Object Exchange” on page 1 of the Palm OS Programmer’s Companion, vol. II, Communications.

SMS Exchange Library Data Structures

SmsParamsType

The SmsParamsType structure identifies information specific to the SMS Exchange Library. The socketRef field of the ExgSocketType structure is set to this structure when you send or receive data using the SMS Exchange Library. You only need to create this structure and assign it to the socketRef field if you have an SMS message to send and want to use non-default values for some of the fields; otherwise, the SMS Exchange Library creates this structure for you and provides default values.

typedef struct SmsParamsTag
{
    UInt32 creator;
    UInt16 smsID;
    Char *extension;
    Char *mimeTypes;
    UInt32 appCreator;
}
SMS Exchange Library
SMS Exchange Library Data Structures

```c
UInt8 dataCodingScheme;
UInt8 networkType;
UInt8 dataType;
UInt16 nbsDestPort;
UInt16 nbsSrcPort;
union
{
    SmsSendParamsType send;
    SmsReceiveParamsType receive;
    SmsReportParamsType report;
}
data;

} SmsParamsType, *SmsParamsPtr;

Field Descriptions

creator
Creator ID of the SMS Exchange Library. Always set this to sysFileCSmsLib.

smsID
The ID of the message that was sent. Do not set this field directly; the SMS Exchange Library should set it.

extension
If the SMS message has an attachment, this field specifies the attachment name. Do not set this field directly; the SMS Exchange Library sets it if necessary. See the appCreator field description for details.

mimeTypes
If the SMS message has an attachment, this field specifies the MIME type of the attachment. Do not set this field directly; the SMS Exchange Library sets it if necessary. See the appCreator field description for details.
appCreator

The creator ID of the target application for the attachment to the SMS message. Do not set this field directly; the SMS Exchange Library sets it if necessary.

When the SMS Exchange Library receives a message with an attachment, it unwraps the message and attempts to deliver the attachment directly to an application that is registered to receive it. If no application is registered to receive unwrapped attachments of that type, the SMS Exchange Library sends the entire SMS message, and it sets the `extension`, `mimeType`, and `appCreator` fields in this structure. The SMS application can use this information to have the Exchange Manager deliver the attachment to the appropriate application using the Local Exchange Library.

dataCodingScheme

The data encoding scheme that the message uses. See SMS Data Coding Scheme Constants.

networkType

Indicates the type of advanced parameters. See SMS Network Type Constants.

dataType

Identifies the type of message being received, such as multipart or return receipt. See SMS Message Type Constants.

nbsDestPort

The Narrow Band Socket (NBS) port on which you want the data sent. The SMS Exchange Library sets this for you if you leave it blank. When data is being received, this field is set to the NBS port on which the data was received.
**SMS Exchange Library**

**SMS Exchange Library Data Structures**

---

**Compatibility**

This structure is only defined if [4.0 New Feature Set](#) is present.

**SmsPrefType**

The `SmsPrefType` structure defines the SMS Exchange Library preferences for sending and receiving SMS messages. Applications can use the `ExgControl` function to get, set, or display these preferences to the user.

```c
typedef struct SmsPrefTag
{
    UInt32 validity;
    UInt16 warnOver;
    Boolean leave;
    Boolean report;
    Boolean autoSMSC;
    Char smscNumber[kSmsMaxPhoneSize];
} SmsPrefType, *SmsPrefPtr;
```
Field Descriptions

validity The number of seconds before the message expires. If the message cannot be delivered to the recipient, the service center repeatedly attempts to deliver the message until it expires. The default is one hour.

warnOver The number of parts a user can send without confirmation. If the user attempts to send a message with more than this number of parts, an alert is displayed, and the user can choose to send the message anyway. The default is 3 parts. (If the user attempts to send a message with more than 3 parts, an alert is displayed.)

leave If true, any incoming messages retrieved from a phone remain on the phone as well. If false, the messages are deleted from the phone’s inbox.

report If true, the user receives confirmation that an SMS message was delivered.

autoSMSC If true, don’t use the value stored in the smscNumber field.

smscNumber The message center to be used. If NULL or the empty string, the SMS message center set by the phone is used.

Compatibility This structure is only defined if 4.0 New Feature Set is present.

SmsReceiveCDMAParamsType

The SmsReceiveParamsType includes an SmsReceiveCDMAParamsType structure for CDMA messages.

typedef struct SmsReceiveCDMAParamsTag
{
    UInt8 messageType;
    TelSmsDateTimeType validityPeriod;
    UInt8 priority;
    UInt8 privacy;
}
Boolean alertOnDeliveryRequest;
Boolean manualAckRequest;
UInt8 voiceMessageNumber;
UInt8 languageIndicator;
Char * callbackNumberAddress;
} SmsReceiveCDMAParamsType,
*SmsReceiveCDMAParamsPtr;

**Field Descriptions**

- **messageType**: The type of the message. This is one of the SMS Message Type Constants constants defined in TelephonyMgr.h.
- **validityPeriod**: The amount of time for which the message is valid. See TelSmsDateTimeType. The default is set according to the SMS preferences.
- **priority**: The message priority. This must be one of the SMS Message Urgency Constants defined in TelephonyMgr.h.
- **privacy**: The privacy type of the message. This must be one of the SMS Message Privacy Constants defined in TelephonyMgr.h.
- **alertOnDeliveryRequest**: true if the user is to be alerted upon delivery of this message, and false if not.
- **manualAckRequest**: true if a confirmation is requested from the recipient, and false if not.
- **voiceMessageNumber**: The number of new messages in your voice mail.
languageIndicator    Reserved for future use.
callbackNumberAddress The callback number to which confirmations are to be sent.

Compatibility   This structure is only defined if 4.0 New Feature Set is present.

SmsReceiveGSMParamsType

The SmsReceiveGSMParamsType includes an SmsReceiveGSMParamsType structure for GSM messages.

typedef struct SmsReceiveGSMParamsTag
{
    UInt16 protocolId;
    Char *serviceCenterNumber;
    Boolean replyPath;
} SmsReceiveGSMParamsType,
*SmsReceiveGSMParamsPtr;

Field Descriptions

protocolId          Reserved for future use.
serviceCenterNumber The SMS service center that must be used to send a reply. If NULL, the service center specified in the preferences is used.
replyPath            If true, replies must be made through the SMS service center specified by serviceCenterNumber.

Compatibility   This structure is only defined if 4.0 New Feature Set is present.

SmsReceiveParamsType

The SmsReceiveParamsType structure is used as the data field for the SmsParamsType structure when the SMS Exchange Library
has received data. The SMS Exchange Library always supplies the values for these fields.

```c
typedef struct SmsReceiveParamsTag
{
    UInt32 timeStamp;
    Char *originatingAddress;
    UInt8 leaveOnPhone:1;
    UInt8 forceSlotMode:1;
    UInt8 reserved:6;
    UInt16 index;
    Boolean otherToReceive;
    Boolean reportDeliveryIndicator;
    union
    {
        SmsReceiveGSMParamsType gsm;
        SmsReceiveCDMAParamsType cdma;
        SmsReceiveTDMAParamsType tdma;
    } protocol;
} SmsReceiveParamsType, *SmsReceiveParamsPtr;
```

**Field Descriptions**

- **timeStamp**
  The time at which the message was delivered, given as the number of seconds since January 1, 1904.

- **originatingAddress**
  The number from which the message was received.

- **leaveOnPhone**
  If true, messages received on the phone are not deleted from the phone’s inbox. If not specified, this is set according to the system preferences.

- **forceSlotMode**
  If true, use slot mode parsing. If false, use block mode parsing. The default is block mode.

- **reserved**
  Reserved for future use.
index

Location where the message is stored on the mobile phone.

otherToReceive

If true, there are more messages to be received from the service center.

reportDeliveryIndicator

If true, the sender has requested confirmation. The recipient of the message does not send the confirmation; the SMS service center does.

protocol

Values specific to the protocol used to send the message. Currently, only GSM is supported.

Compatibility

This structure is only defined if 4.0 New Feature Set is present.

**SmsReceiveTDMAParamsType**

The SmsReceiveParamsType includes an SmsReceiveTDMAParamsType structure for TDMA messages. This structure is currently the same as the SmsReceiveCDMAParamsType structure.

```c
typedef SmsReceiveCDMAParamsType SmsReceiveTDMAParamsType,
*SmsReceiveTDMAParamsPtr;
```

Compatibility

This structure is only defined if 4.0 New Feature Set is present.

**SmsReportParamsType**

The SMSReportParamsType structure is used as the data field for the SmsParamsType structure when the SMS Exchange Library has received a delivery confirmation. The SMS Exchange Library always sets the values for these fields.

```c
typedef struct SmsReportParamsTag {
    UInt32 timeStamp;
```
SmsReportParamsType, *SmsReportParamsPtr;

Field Descriptions

timeStamp
The date and time at which the message was delivered, given as the number of seconds since January 1, 1904.

index
Location where the message is stored on the mobile phone.

reportType
One of the Delivery Report Type constants defined in TelephonyMgr.h.

report
One of the Delivery Status Report constants defined in TelephonyMgr.h.

originatingAddress
Phone number to which the message was sent.

Compatibility
This structure is only defined if 4.0 New Feature Set is present.

SmsSendCDMAParamsType
The SmsSendCDMAParamsType includes an SmsSendCDMAParamsType structure for CDMA messages.

typedef struct SmsSendCDMAParamsTag
{
    UInt8   messageType;
    TelSmsDateTimeType deferredDate;
    UInt8   priority;
    UInt8   privacy;
    UInt8   alertOnDelivery:1;
    UInt8   manualAckRequest:1;
    UInt8   reserved:6;
    Char*   callbackNumber;
} SmsSendCDMAParamsType, *SmsSendCDMAParamsPtr;
Field Descriptions

messageType

The type of the message. This is one of the SMS Message Type Constants constants defined in TelephonyMgr.h.

deferedDate

Not used.
priority

The message priority. This must be one of the SMS Message Urgency Constants.

privacy

The privacy type of the message. This must be one of the SMS Message Privacy Constants.

alertOnDelivery

ture if the user is to be alerted upon delivery of this message, and false if not.

manualAckRequest

ture if a confirmation is requested from the recipient, and false if not.

reserved

Reserved for future use.
callbackNumber

Number to which the confirmation should be sent.

Compatibility

This structure is only defined if 4.0 New Feature Set is present.

SmsSendGSMParamsType

The SmsSendGSMParamsType includes an SmsSendGSMParamsType structure for GSM messages.

typedef struct SmsSendGSMParamsTag
{
    UInt16 protocolId;
    Char *serviceCenterNumber;
    Boolean rejectDuplicated;
    Boolean replyPath;
} SmsSendGSMParamsType, *SmsSendGSMParamsPtr;
Field Descriptions

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocolId</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>serviceCenterNumber</td>
<td>The message center to be used. If not specified, the service center is set according to the system preferences.</td>
</tr>
<tr>
<td>rejectDuplicated</td>
<td>If true, the service center rejects messages that have the same message ID, destination address, and originating address as a previously submitted message.</td>
</tr>
<tr>
<td>replyPath</td>
<td>If true, the service center that delivers the message is requested to provide information about itself to the recipient so that replies are made through the same service center.</td>
</tr>
</tbody>
</table>

Compatibility

This structure is only defined if 4.0 New Feature Set is present.

SMSSendParamsType

The SMSSendParamsType structure is used as the data field for the SmsParamsType structure when the SMS Exchange Library is sending data.

typedef struct SmsSendParamsTag
{
    TelSmsDateTimeType validityPeriod;
    Char *destinationAddress;
    UInt8 networkDeliveryRequested:1;
    UInt8 ignoreDefaultValue:1;
    UInt8 reserved:6;
    UInt16 partCount;
    UInt16 lastPart;
    UInt8 converter;
    union
    {
        SmsSendGSMParamsType gsm;
        SmsSendCDMAParamsType cdma;
    }
}
```c
SmsSendTDMAParamsType tdma;
} protocol;
} SmsSendParamsType, *SmsSendParamsPtr;

**Field Descriptions**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>validityPeriod</strong></td>
<td>The amount of time for which the message is valid. See TelSmsDateTimeType. The default is set according to the SMS preferences.</td>
</tr>
<tr>
<td><strong>destinationAddress</strong></td>
<td>A buffer that contains the phone number of the message recipient. If no phone number is supplied, the user is prompted for the phone number.</td>
</tr>
<tr>
<td><strong>networkDeliveryRequested</strong></td>
<td>If true, the SMS service center sends a delivery confirmation. The default is set according to the SMS preferences. The SMS Exchange Library disables this field for multipart messages.</td>
</tr>
<tr>
<td><strong>ignoreDefaultValue</strong></td>
<td>If false, the validity period, network delivery requested, and SMS center specified in the preferences are used regardless of the values supplied in this structure. If true, the values supplied in this structure are used.</td>
</tr>
<tr>
<td><strong>reserved</strong></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td><strong>partCount</strong></td>
<td>The number of parts in the message. 0 means that the message is not a multipart message. If NBS is used to send the message, it determines the number of parts.</td>
</tr>
</tbody>
</table>
lastPart  The last part of a multipart message that was successfully sent.

converter  The header added to the data to specify how it is converted.

protocol  Data specific to the protocol used to send the message. Currently, only GSM is supported.

Compatibility  This structure is only defined if 4.0 New Feature Set is present.

**SmsSendTDMAParamsType**

The *SmsSendParamsType* includes an *SmsSendTDMAParamsType* structure for TDMA messages. This structure is currently the same as the *SmsSendCDMAParamsType* structure.

```c
typedef SmsSendCDMAParamsType
    SmsSendTDMAParamsType,
    *SmsSendTDMAParamsPtr;
```

Compatibility  This structure is only defined if 4.0 New Feature Set is present.

**SMS Exchange Library Constants**

**SMS Control Constants**

The SMS control constants are passed as the operation parameter to the *ExgControl* function. The *ExgControl* function is a way to communicate directly with the SMS Exchange Library. The following table lists the operation constant, the type of data that should be passed as the *valueP* parameter to *ExgControl*, and what the SMS Exchange Library does in response.
### Table 84.1 ExgControl operations for SMS library

<table>
<thead>
<tr>
<th>Operation</th>
<th>value Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exgLibSmsPrefGetOp</td>
<td>SmsPrefType</td>
<td>Returns a pointer to the SMS Exchange Libraries preferences in valueP, creating the preferences and setting them to the default values if they do not exist.</td>
</tr>
<tr>
<td>exgLibSmsPrefGetDefaultOp</td>
<td>SmsPrefType</td>
<td>Returns the default values for the SMS Exchange Library preferences.</td>
</tr>
<tr>
<td>exgLibSmsPrefSetOp</td>
<td>SmsPrefType</td>
<td>Sets the SMS Exchange Library preferences to the values passed in valueP.</td>
</tr>
<tr>
<td>exgLibSmsPrefDisplayOp</td>
<td>kSmsNetworkAuto or kSmsNetworkGSM</td>
<td>Input only. Display a form that allows the user to set the SMS preferences.</td>
</tr>
<tr>
<td>exgLibSmsIncompleteGetCountOp</td>
<td>UInt16</td>
<td>Output only. Get the number of incomplete messages currently stored in the SMS Exchange Library. The library stores message parts as it receives them. When it has received all of the parts, it reassembles the message and delivers it. This operation tells how many messages are currently under assembly.</td>
</tr>
<tr>
<td>exgLibSmsIncompleteDeleteOp</td>
<td>UInt16</td>
<td>Input only. Delete the incomplete message with the ID passed in valueP. Pass -1 to delete all incomplete messages.</td>
</tr>
</tbody>
</table>

**Compatibility** These constants are only defined if 4.0 New Feature Set is present.
SMS Data Coding Scheme Constants

The SMS data coding scheme constants describe the coding scheme used for SMS data. These values are used as the `dataCodingScheme` parameter of the `SmsParamsType` structure.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kSmsRowDataEncoding</td>
<td>0</td>
<td>8-bit encoding scheme. This is the default.</td>
</tr>
<tr>
<td>kSmsTextEncoding</td>
<td>1</td>
<td>7-bit encoding scheme.</td>
</tr>
</tbody>
</table>

**Compatibility**

These constants are only defined if 4.0 New Feature Set is present.

SMS Network Type Constants

The SMS network type constants identify the type of network being used for SMS messages. Currently, only the GSM network is supported.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kSmsNetworkAuto</td>
<td>-1</td>
<td>The network is set by the phone. This is the default.</td>
</tr>
<tr>
<td>kSmsNetworkCDMA</td>
<td>kTelNwkCDMA</td>
<td>A CDMA network. Currently not supported.</td>
</tr>
<tr>
<td>kSmsNetworkGSM</td>
<td>kTelNwkGSM</td>
<td>A GSM network.</td>
</tr>
<tr>
<td>kSmsNetworkTDMA</td>
<td>kTelNwkTDMA</td>
<td>A TDMA network. Currently not supported.</td>
</tr>
<tr>
<td>kSmsNetworkPDC</td>
<td>kTelNwkPDC</td>
<td>A PDC network. Currently not supported.</td>
</tr>
</tbody>
</table>

**Compatibility**

These constants are only defined if 4.0 New Feature Set is present.

SMS Message Type Constants

The SMS message type constants identify the type of message being sent. They are used as the `dataType` field of the `SmsParamsType` structure.
### SMS Exchange Library

#### SMS Exchange Library Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kSmsMessageType</td>
<td>0</td>
<td>Standard SMS message of no more than 160 bytes. This is the default.</td>
</tr>
<tr>
<td>kSmsIncompleteType</td>
<td>1</td>
<td>A part of a multipart SMS message.</td>
</tr>
<tr>
<td>kSmsReportType</td>
<td>2</td>
<td>A confirmation, indicating that an SMS message was successfully sent.</td>
</tr>
</tbody>
</table>

**Compatibility** These constants are only defined if [4.0 New Feature Set](#) is present.

---

### SMS Converter Constants

The SMS converter constants identify the header information added to an SMS message.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kSmsNBSConverter</td>
<td>0</td>
<td>An NBS header is added to the message. This is the default.</td>
</tr>
<tr>
<td>kSmsNoConverter</td>
<td>1</td>
<td>No header is added to the message.</td>
</tr>
</tbody>
</table>

**Compatibility** These constants are only defined if [4.0 New Feature Set](#) is present.
SMS Exchange Library
SMS Exchange Library Constants
Personal Data Interchange Library

This chapter provides reference material for the Personal Data Interchange (PDI) library, which provides tools for reading and writing vObjects, including vCards and vCals. This chapter discusses the following topics:

- PDI Library Data Structures
- PDI Library Constants
- PDI Library Functions

The header file `PdiLib.h` declares the Personal Data Interchange library API. The header file `PdiConst.h` declares the constants that you use with the PDI library.

For information about how to use the functions and constants described in this chapter, see Chapter 3, “Personal Data Interchange,” in Palm OS Programmer’s Companion, vol. II, Communications.

PDI Library Data Structures

This section describes the data structures used with the PDI library functions.

PdiDictionary

The `PdiDictionary` type is a simple typedef that represents an internal, binary object.
typedef UInt8 PdiDictionary;

PdiReaderType
The PdiReaderType data structure represents a PDI reader object, which you use to read data from an input stream.

typedef struct _PdiReader {
    Err error;
    Uint8 encoding;
    CharEncodingType charset;
    Uint16 written;
    Uint8 fieldNum;
    Uint16 property;
    Uint16 propertyValueType;
    Uint16 parameter;
    Uint32 parameterPairs[kPDI_ENTRIES_NUMBER];
    Uint16 customFieldNumber;
    void *appData;
    Uint16 pdiRefNum;
    Uint16 events;
    Char *groupName;
    Char *propertyName;
    Char *parameterName;
    Char *parameterValue;
    Char *propertyValue;
    PdiDictionary *dictionary[2];
} PdiReaderType

Field Descriptions

error The most recent error.
encoding The type of encoding for the property value.
charset The character set of the property value.
written The number of characters that have currently been written to the buffer.
fieldNum The current field number.
property

The ID of the current property.

propertyValueType

The value type of the current property value.

parameter

The ID of the most recently parsed parameter name.

parameterPairs

An integer array with bits set for each parameter value that has been parsed for the current property value.

NOTE: You must use the PdiParameterPairTest macro to access this field.

customFieldNumber

The number of the custom field parsed by the reader for the current property. Custom fields are used in the Palm™ Address Book.

appData

Application-dependent data field.

pdiRefNum

The library reference number associated with this reader.

events

The mask of events handled by the reader in its most recent operation. This is a combination of some number of the event constants described in Reader Event Constants.

groupName

The group name for the current property.

propertyName

The name of the current property.

parameterName

The name of the current parameter.

parameterValue

The value of the current parameter.
The `PdiWriterType` data structure represents a PDI writer object, which you use to write data to an output stream.

typedef struct _PdiWriter {
    void               *appData;
    UInt16             pdiRefNum;
    UInt16             encoding;
    CharEncodingType   charset;
    Err                error;
    PdiDictionary      *dictionary[2];
} PdiWriterType

**Field Descriptions**

- **appData** Application-dependent data field.
- **pdiRefNum** The library reference number associated with this reader.
- **encoding** The type of encoding for the property value.
- **charset** The character set of the property value.
PDI Library Constants

This section describes the constants used in the PDI library, which include the following constant types:

- Buffer Management Constants
- Encoding Type Constants
- Error Code Constants
- Parameter Name Constants
- Parameter Value Constants
- Property Name Constants
- Property Type Constants
- Property Value Field Constants
- Property Value Format Constants
- Reader and Writer Options Constants
- Reader Event Constants
- Value Type Constants

Buffer Management Constants

You use the buffer management constants to determine how buffers are managed in the PDI reader.

error

The most recent error.

dictionary

An array of two dictionary pointers. The dictionary[0] entry is the main dictionary built into the PDI library, and the dictionary[1] entry is an optional, custom dictionary associated with the reader object with a call to PdiDefineReaderDictionary.
Encoding Type Constants

You use the encoding type constants to specify the type of encoding used in a vObject reader or writer.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kPdiASCIIEncoding</td>
<td>0</td>
<td>The vObject is not encoded.</td>
</tr>
<tr>
<td>kPdiQPEncoding</td>
<td>kPdiPAV_ENCODING_QUOTED_PRINTABLE</td>
<td>The vObject uses the quoted printable encoding.</td>
</tr>
<tr>
<td>kPdiB64Encoding</td>
<td>kPdiPAV_ENCODING_BASE64</td>
<td>The vObject uses Base 64 encoding. The writer outputs &quot;ENCODING=BASE64.&quot;</td>
</tr>
<tr>
<td>kPdiBEncoding</td>
<td>kPdiPAV_ENCODING_B</td>
<td>The vObject uses Base 64 encoding. This is the same as the kPdiB64Encoding value, except that the PDI writer outputs &quot;ENCODING=B.&quot; This encoding is used with the vCard 3.0 standard.</td>
</tr>
</tbody>
</table>
The PDI library functions return the error code constants shown in the following table to indicate their status.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kPdiEscapeEncoding</td>
<td>0x8000</td>
<td>The vObject uses escapes for special characters.</td>
</tr>
<tr>
<td>kPdiNoEncoding</td>
<td>0x8001</td>
<td>The PDI writer does not encode the vObject value.</td>
</tr>
</tbody>
</table>

### Error Code Constants

The PDI library functions return the error code constants shown in the following table to indicate their status.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pdiErrRead</td>
<td>An error occurred while reading from the input stream.</td>
</tr>
<tr>
<td>pdiErrWrite</td>
<td>An error occurred while writing to the output stream.</td>
</tr>
<tr>
<td>pdiErrNoPropertyName</td>
<td>An attempt was made to write a property value before the property name was written.</td>
</tr>
<tr>
<td>pdiErrNoPropertyValue</td>
<td>The application did not write the last property value.</td>
</tr>
<tr>
<td>pdiErrMoreChars</td>
<td>The buffer is full. Superfluous characters have been discarded.</td>
</tr>
<tr>
<td>pdiErrNoMoreFields</td>
<td>There are no more property fields to read.</td>
</tr>
<tr>
<td>pdiErrOpenFailed</td>
<td>The PDI library could not be opened.</td>
</tr>
<tr>
<td>pdiErrCloseFailed</td>
<td>The PDI library could not be closed. This can occur if another application is using the library.</td>
</tr>
</tbody>
</table>
Parameter Name Constants

The PdiConst.h file defines several parameter name constants that you can use to specify the name of a parameter in functions that use parameter names. The parameter name constants have the following format:

\[ \text{kPdiPAN}_{\text{parameterName}} \]

where \text{parameterValue} is replaced by a parameter name.

The following table shows examples of parameter name constants. For a complete list, see the PdiConst.h file.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kPdiPAN_TYPE</td>
<td>The TYPE parameter.</td>
</tr>
<tr>
<td>kPdiPAN.Encoding</td>
<td>The ENCODING parameter.</td>
</tr>
<tr>
<td>kPdiPAN_STATUS</td>
<td>The STATUS parameter.</td>
</tr>
</tbody>
</table>

Parameter Value Constants

The PdiConst.h file defines several parameter value constants that you can use to specify the name and value of a parameter in functions that use name and value pairs. The parameter value constants have the following format:

\[ \text{kPdiPAV}_{\text{parameterName}}_{\text{parameterValue}} \]

where \text{parameterName} is replaced by a parameter name and \text{parameterValue} is replaced by a parameter value.

The following table shows examples of parameter value constants. For a complete list, see the PdiConst.h file.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kPdiPAV_TYPE_TEL</td>
<td>The parameter name is TYPE and the parameter value is TEL.</td>
</tr>
<tr>
<td>kPdiPAV.Encoding_BASE64</td>
<td>The parameter name is ENCODING and parameter value is BASE64.</td>
</tr>
</tbody>
</table>
The PdiCons.h file defines several property name constants that you can use to specify the name of a PDI property in functions that use property names. The property name constants have the following format:

\[ \text{kPdiPRN}_\text{propertyName} \]

where \( \text{propertyName} \) is replaced by a property name.

The following table shows examples of property name constants. For a complete list, see the PdiCons.h file.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kPdiPRN_ADR</td>
<td>The ADR property.</td>
</tr>
<tr>
<td>kPdiPRN_BDAY</td>
<td>The BDAY property.</td>
</tr>
<tr>
<td>kPdiPRN_BEGIN</td>
<td>The BEGIN property.</td>
</tr>
<tr>
<td>kPdiPRN_BEGIN_VCARD</td>
<td>The BEGIN:VCARD property.</td>
</tr>
</tbody>
</table>

### Property Type Constants

You use the property type constants to specify the data type of a property.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kPdiType_URI</td>
<td>0</td>
<td>The data is a uniform resource identifier.</td>
</tr>
<tr>
<td>kPdiType.UTC_OFFSET</td>
<td>1</td>
<td>The data is an offset from UTC to local time.</td>
</tr>
</tbody>
</table>
Property Value Field Constants

The PdiConst.h file defines several property value field constants that you can use to specify the position of a PDI property value field in functions that use fields. The property value field constants have the following format:

    kPdiPVF_propertyValueField

where propertyValueField is replaced by a property value field name.

The following table shows examples of property name constants. For a complete list, see the PdiConst.h file.
Some properties have structured values, which are values that contain multiple fields. These fields are typically separated by commas or semicolons in the vObject input or output stream. You use the property value format constants with the `PdiReadPropertyField` and `PdiWritePropertyStr` functions to specify how to handle fields in a structured value.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kPdiNoFields</td>
<td>0</td>
<td>There are no fields in the property value; <code>PdiReadPropertyField</code> reads the entire value, or <code>PdiWritePropertyStr</code> specifies that the entire value should be written.</td>
</tr>
<tr>
<td>kPdiCommaFields</td>
<td>1</td>
<td>Fields are separated with comma (&quot;,&quot;) characters; <code>PdiReadPropertyField</code> reads one field, or <code>PdiWritePropertyStr</code> specifies that one field should be written.</td>
</tr>
</tbody>
</table>
Reader and Writer Options Constants

You use the reader and writer option constants to control how the PDI reader (parser) reads values from the input stream or to control how the PDI writer (generator) writes values to the output stream.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kPdiSemicolonFields</td>
<td>2</td>
<td>Fields are separated with semicolon (&quot;;&quot;) characters; PdiReadPropertyField reads one field, or PdiWritePropertyStr specifies that one field should be written.</td>
</tr>
<tr>
<td>kPdiDefaultFields</td>
<td>4</td>
<td>The parser decides the property value format, based on the property name.</td>
</tr>
<tr>
<td>kPdiConvertComma</td>
<td>8</td>
<td>Fields are separated with comma characters; PdiReadPropertyField reads the entire value and converts each comma into a newline (&quot;\n&quot;) character.</td>
</tr>
<tr>
<td>kPdiConvertSemicolon</td>
<td>16</td>
<td>Fields are separated with semicolon characters; PdiReadPropertyField reads the entire value and converts each semicolon into a newline (&quot;\n&quot;) character.</td>
</tr>
</tbody>
</table>
### PDI Library Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kPdiEnableFolding</td>
<td>1</td>
<td>Enables folding of properties in the output stream.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Folding</strong> is a mechanism for breaking long lines to allow them to be transmitted without change. If you specify this flag, the PDI library folds long lines. Note that folding is not compatible with earlier versions of the Palm OS®. Also note that other encoding formats, including quoted-printable and Base 64, define their own mechanisms for splitting long lines.</td>
</tr>
<tr>
<td>kPdiEnableQuotedPrintable</td>
<td>2</td>
<td>Enables quoted-printable encoding in the output stream and makes it the default encoding. This is an encoding format for non-ASCII values. You must have this enabled for compatibility with earlier versions of the Palm OS. If you do not specify this option, the default encoding is Base 64.</td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>kPdiEscapeMultiFieldValues</td>
<td>4</td>
<td>For compatibility with earlier versions of the Palm OS. You must enable this for compatibility with earlier versions of the Palm OS. However, some non-Palm PDI software does not support this format. For more information about compatibility with earlier versions of the Palm OS, see Format Compatibility in the Palm OS Programmer’s Companion, vol. II, Communications.</td>
</tr>
<tr>
<td>kPdiPalmCompatibility</td>
<td>6</td>
<td>This is a combination of kPdiEnableQuotedPrintable</td>
</tr>
<tr>
<td>kPdiEnableB</td>
<td>8</td>
<td>Enables base 64 encoding in the output stream, and tells the PDI writer to output &quot;ENCODING=B&quot; instead of &quot;ENCODING=BASE64&quot; when encoding a value with Base 64. Note: the vCard 3.0 standard has replaced the earlier ENCODING=BASE64 with ENCODING=B. The meaning is the same.</td>
</tr>
<tr>
<td>kPdiOpenParser</td>
<td>8</td>
<td>Specifies that the PDI reader is open to all formats, including Palm and others.</td>
</tr>
</tbody>
</table>
Reader Event Constants

The PDI reader event constants specify the events that the reader has handled during the current read operation. The event values are combined together and stored in the `events` field of the PDI reader object. You can use them to test whether the reader handled a certain event.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kPdiEOFEventMask</td>
<td>1</td>
<td>End of file was reached.</td>
</tr>
<tr>
<td>kPdiGroupNameEventMask</td>
<td>2</td>
<td>A group name was found.</td>
</tr>
<tr>
<td>kPdiPropertyNameEventMask</td>
<td>4</td>
<td>A property name was found.</td>
</tr>
<tr>
<td>kPdiParameterNameEventMask</td>
<td>8</td>
<td>A parameter name was found.</td>
</tr>
<tr>
<td>kPdiParameterValueEventMask</td>
<td>16</td>
<td>A parameter value was found.</td>
</tr>
<tr>
<td>kPdiPropertyDefinedEventMask</td>
<td>32</td>
<td>A property definition was found; this implies that the &quot;:&quot; separator character was found.</td>
</tr>
<tr>
<td>kPdiPropertyValueEventMask</td>
<td>64</td>
<td>An entire property value was found</td>
</tr>
<tr>
<td>kPdiPropertyValueFieldEventMask</td>
<td>128</td>
<td>A value field was found; this implies that the &quot;;&quot; separator character was found.</td>
</tr>
<tr>
<td>kPdiPropertyValueItemEventMask</td>
<td>256</td>
<td>A value item was found; this implies that the &quot;,&quot; separator character was found.</td>
</tr>
</tbody>
</table>
Value Type Constants

You can use the following constants to specify data typing information for the `PdiWritePropertyBinaryValue`, `PdiWritePropertyFields`, and `PdiWritePropertyValue` functions.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kPdiWriteData</td>
<td>0</td>
<td>The value is data. The PDI writer does not compute a character set. You can use this for binary data or pure ASCII data.</td>
</tr>
<tr>
<td>kPdiWriteText</td>
<td>8</td>
<td>The value is text data. The PDI writer parses the data character by character to compute the correct charset and character encoding for the data.</td>
</tr>
<tr>
<td>kPdiWriteMultiline</td>
<td>16</td>
<td>Explicitly specifies that the value contains special characters, such as newlines, and must be encoded. If this flag is not specified, the encoding is determined by the applied character set.</td>
</tr>
</tbody>
</table>
PDI Library Functions

PdiDefineReaderDictionary

Purpose
Installs a new custom dictionary for use with a PDI reader object.

Declared In
PdiLib.h

Prototype
PdiDictionary *PdiDefineReaderDictionary
(UInt16 libRefnum, PdiReaderType *ioReader,
PdiDictionary *dictionary,
Boolean disableMainDictionary)

Parameters
-> libRefnum      The PDI library reference number.
-> ioReader       The PDI reader object with which to associate the dictionary. This object must have previously been created by a call to the PdiReaderNew function.
-> dictionary    A pointer to a dictionary object that was created by the . The dictionary object is an array of binary data.
-> disableMainDictionary
                 If true, the main reader dictionary is disabled, and only this new dictionary is searched for terms; if false, the new dictionary supplements the main dictionary.

Result
Returns a pointer to the previously installed custom dictionary, or NULL if there was not a previously installed custom dictionary.

Comments
This function installs a dictionary for use with the ioReader object. The dictionary contains the syntax for extensions or replacements of the PDI properties about which the PDI reader knows. The reader knows about properties specified in one of the vObject standards, including the vCard or vCal standards.

You can uninstall the current custom dictionary by specifying NULL as the value of the dictionary parameter,
Note that the dictionary must have previously been compiled by the dictionary tool. For more information, review the PDI sample code, which you can find on the Palm Developer’s Knowledge Base at http://www.palmos.com/dev/tech/kb.

**PdiDefineResizing**

**Purpose**  Defines the sizing information to use when automatically resizing a buffer. PDI reader objects read information from the input stream into a buffer and automatically adjust the buffer size as required.

**Declared In**  PdiLib.h

**Prototype**  
```
Err PdiDefineResizing(UInt16 libRefnum,
PdiReaderType *ioReader, UInt16 deltaSize,
UInt16 maxSize)
```

**Parameters**

- `libRefnum`  The PDI library reference number.
- `ioReader`  The PDI reader object, which was created by a previous call to the `PdiReaderNew` function.
- `deltaSize`  The number of bytes by which to grow the buffer when it needs resizing.
- `maxSize`  The maximum allowable size for the buffer.

**Result**  Returns `errNone` if successful, and an error code if not successful.

**Comments**  This function redefines the values to use when resizing a buffer. It does not perform any other actions.

The resizing values are used if your reader runs out of buffer space when storing input data during the processing of a property value. If possible, the reader resizes its internal buffer, using the values that you supply in this function.

The default resizing values apply if you do not call this function. The default values are:

- `kPdiDefaultBufferDeltaSize` 0x0010
- `kPdiDefaultBufferMaxSize` 0x3FFF
**Compatibility**  Implemented only if 4.0 New Feature Set is present.

**PdiDefineWriterDictionary**

**Purpose**  Installs a new custom dictionary for use with a PDI writer object.

**Declared In**  PdiLib.h

**Prototype**

```
PdiDictionary *PdiDefineWriterDictionary
(UInt16 libRefnum, PdiWriterType *ioWriter,
PdiDictionary *dictionary,
Boolean disableMainDictionary)
```

**Parameters**

- `libRefnum`  The PDI library reference number.
- `ioWriter`  The PDI writer object with which to associate the dictionary. This object must have previously been created by a call to the PdiWriterNew function.
- `dictionary`  A pointer to a dictionary object that was created by the . The dictionary object is an array of binary data.
- `disableMainDictionary`  If true, the main dictionary is disabled, and only this new dictionary is searched for terms; if false, the new dictionary supplements the main dictionary.

**Result**  Returns a pointer to the previously installed custom dictionary, or NULL if there was not a previously installed custom dictionary.

**Comments**  This function installs a dictionary for use with the ioWriter object. The dictionary contains the syntax for extensions or replacements of the PDI properties about which the PDI writer knows. The writer knows about properties specified in one of the vObject standards, including the vCard or vCal standards.

You can uninstall the current custom dictionary by specifying NULL as the value of the dictionary parameter,
Note that the dictionary must have previously been compiled by the dictionary tool. For more information, review the PDI sample code, which you can find on the Palm Developer’s Knowledge Base at http://www.palmos.com/dev-tech/kb/.

### PdiEnterObject

**Purpose**

Tells the PDI library to enter into a recursively-defined object.

**Declared In**

PdiLib.h

**Prototype**

Err PdiEnterObject(UInt16 libRefnum, PdiReaderType *ioReader)

**Parameters**

- `libRefnum`  The PDI library reference number.
- `ioReader`   The PDI reader object, which was created by a previous call to the PdiReaderNew function.

**Result**

Returns errNone if successful, and an error code if not successful.

**Comments**

Some vObjects recursively define other vObjects. Your application can choose whether or not to enter and parse the recursively defined objects.

If you want to parse the nested object definition, you need to call this function; otherwise, all of the properties of the nested object are skipped when the next call is made to the PdiReadProperty or PdiReadPropertyName functions.

Call this function after a BEGIN_VObject statement of the nested object has been parsed.

**Compatibility**

Implemented only if 4.0 New Feature Set is present.
PdiLibClose

**Purpose**
Close the PDI library after your application has finished with it.

**Declared In**
PdiLib.h

**Prototype**
Err PdiLibClose(UInt16 libRefnum)

**Parameters**
- `libRefnum` The PDI library reference number.

**Result**
Returns 0 if no other application uses the library. You may need to call [SysLibRemove](#) to remove the PDI library from the system library table.

Returns `pdiErrCloseFailed` if the library could not be closed.

**Comments**
You should call this function after your application has finished with the PDI library, to allow the resources to be recovered.

**Compatibility**
Implemented only if [4.0 New Feature Set](#) is present.

**See Also**
PdiLibOpen

PdiLibOpen

**Purpose**
Opens the PDI library for use by your application.

**Declared In**
PdiLib.h

**Prototype**
Err PdiLibOpen(UInt16 libRefnum)

**Parameters**
- `libRefnum` The PDI library reference number.

**Result**
Returns `errNone` if successful, and an error code if not successful.

**Comments**
You must call this function before calling any of the other PDI functions.
PdiParameterPairTest

Purpose
A macro that determines if the reader has already parsed the specified parameter value or name-value pair.

Declared In
PdiLib.h

Prototype
PdiParameterPairTest (reader, pair)

Parameters
- `reader` The PDI reader object, which was created by a previous call to the PdiReaderNew function.
- `pair` The ID of the parameter. This must be one of the Parameter Value Constants.

Result
Returns true if the specified parameter name-value pair has been parsed for the current property, and false if not.

Comments
Some vObject generators do not specify the parameter name if the name is considered evident from the context. This means that both of the following constructs are considered proper:

```
Name=Value
Value
```

The PdiParameterPairTest macro returns true if the value has been parsed in either format. For example,

```
PdiParameterPairTest(reader, kPdiPAV_TYPE_WORK)
```
returns true for either of the following:

```
Type=WORK
WORK
```

Compatibility
Implemented only if 4.0 New Feature Set is present.
PdiReaderDelete

**Purpose** Delete a PDI reader object that is associated with the specified library number.

**Declared In** PdiLib.h

**Prototype**

```c
void PdiReaderDelete(UInt16 libRefnum, PdiReaderType **ioReader)
```

**Parameters**

- `libRefnum` The PDI library reference number.
- `ioReader` A pointer to the PDI reader object, which was created by a previous call to the PdiReaderNew function.

**Result** Returns nothing.

**Comments** This function deletes the UDAReader object associated with the reader object and frees the memory that was allocated for the reader object. The `ioReader` parameter is set to NULL.

**Compatibility** Implemented only if 4.0 New Feature Set is present.

**See Also** PdiReaderNew

PdiReaderNew

**Purpose** Create and initialize a new PDI reader object for use with the specified PDI library number.

**Declared In** PdiLib.h

**Prototype**

```c
PdiReaderType *PdiReaderNew(UInt16 libRefnum, UDAReaderType *input, UInt16 version)
```

**Parameters**

- `libRefnum` The PDI library reference number.
The Unified Data Access (UDA) input object associated with the reader.

Options to control the parsing behavior of the reader. You can use a combination of the Reader and Writer Options Constants.

Returns a pointer to the new PDI reader object. Returns NULL if the reader cannot be created.

The current implementation of the PdiReaderNew function does not make use of the optionFlags settings because the reader knows how to adapt itself to all of the supported formats. The options will be used in future versions.

The input value is a UDA object for reading data from an input stream that can be connected to various data sources. For example, you can use a UDAExchangeReader to read data from the Exchange Manager, and you can use a UDASTringReader to read data from a string. For more information about the UDA Manager, see Chapter 86, “Unified Data Access Manager.”

Implemented only if 4.0 New Feature Set is present.

PdiReaderDelete, PdiWriterNew

Purpose
Read a single parameter name and its value from an input stream.

Declared In PdiLib.h

Err PdiReadParameter(UInt16 libRefnum, PdiReaderType *ioReader)

The PDI library reference number.
Palm OS Programmer's API Reference

PdiReadProperty

Purpose
Read the next property and its parameters from the input stream.

Declared In
PdiLib.h

Prototype
Err PdiReadProperty(UInt16 libRefnum, PdiReaderType *ioReader)

Parameters
-> libRefnum The PDI library reference number.
-> ioReader The PDI reader object, which was created by a previous call to the PdiReaderNew function.

Result
Returns errNone if successful. Returns kPdiReadError if an error occurs.

Comments
The PdiReadProperty function reads a property name and its parameters, by reading until it encounters the PDI "":"" separator character.
This function looks each name up in the properties dictionary, and sets the appropriate bit in the ioReader object structure to indicate that property-parameter pair has been read. The properties dictionary stores information about properties that are considered well known, as described in The PDI Library Properties Dictionary in Chapter 3, “Personal Data Interchange,” in Palm OS Programmer’s Companion, vol. II, Communications.

To read a property, you call PdiReadProperty, followed by a call or calls to the PdiReadPropertyField function to read the property value. For more information, see Reading Properties in Chapter 3, “Personal Data Interchange,” in Palm OS Programmer’s Companion, vol. II, Communications.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

PdiReaderNew, PdiReadPropertyField, PdiReadPropertyName, PdiReadParameter

PdiReadPropertyField

Purpose

Read one field of a property value. The property value can be structured to contain multiple fields that are separated by commas or semicolons.

Declared In

PdiLib.h

Prototype

Err PdiReadPropertyField(UInt16 libRefnum, PdiReaderType *ioReader, Char **bufferPP, UInt16 bufferSize, UInt16 readMode)

Parameters

- libRefnum The PDI library reference number.
- ioReader The PDI reader object, which was created by a previous call to the PdiReaderNew function.
- bufferPP A pointer to a pointer to the buffer into which the field characters are stored. Set this value to NULL to allow the PDI library to manage it.
Note that the PDI library may need to resize the buffer; thus, the value of this parameter might change.

-> bufferSize  The size, in bytes, of the input buffer for reading the field.

You can use the PdiResizableBuffer constant to specify that the PDI Library can automatically resize the buffer as required.

If you do not specify the PdiResizableBuffer value, then the PDI library assumes that buffer cannot be moved, and that its size is fixed.

-> readMode  The format of the fields in the property value. Use one of the Property Value Format Constants.

Result 0  The field was read successfully.

kPdiNoMoreFieldsError  There are no more fields to read because the entire value has already been read.

kPdiMoreCharsError  The buffer is not large enough to store the entire field.

Comments  The value returned in the buffer is terminated with the “\0” character.

If the field is an empty string, the buffer is erased from memory, and the value of buffer is set to NULL.

If you specify kPdiResizableBuffer for the value of the bufferSize parameter, and the buffer needs more space, PdiReadPropertyField resizes the buffer for you, which may cause the value of buffer to be modified.

This function initializes the propertyValue and fieldNum fields of the ioReader object.

To read a property, you usually call the PdiReadProperty function, followed by a call or calls to PdiReadPropertyField to
read the property value. For more information, see Reading Properties in Chapter 3, “Personal Data Interchange,” in Palm OS Programmer's Companion, vol. II, Communications.

Compatibility
Implemented only if 4.0 New Feature Set is present.

See Also PdiReaderNew, PdiReadProperty, PdiReadPropertyName, PdiReadParameter

PdiReadPropertyName

Purpose Read a property name from an input stream. Use this function when you want to parse and process each parameter individually.

Declared In PdiLib.h

Prototype Err PdiReadPropertyName(UInt16 libRefnum, PdiReaderType *ioReader)

Parameters
- -> libRefnum The PDI library reference number.
- -> ioReader The PDI reader object, which was created by a previous call to the PdiReaderNew function.

Result Returns errNone if successful, and an error code if not successful.

Comments The PdiReadProperty function reads a property name only, reading until it encounters the PDI “:” separator character, or until it encounters the first parameter “,” separator character.

This function initializes the property and propertyName fields of the ioReader object.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also PdiReaderNew, PdiReadPropertyField, PdiReadProperty, PdiReadParameter
**PdiSetCharset**

**Purpose**  
Force the character set of the next property value that is written by the specified PDI writer.

**Declared In**  
PdiLib.h

**Prototype**  
Err PdiSetCharset(UInt16 libRefnum, PdiWriterType *ioWriter, CharEncodingType charset)

**Parameters**
- `-> libRefnum`  
The PDI library reference number.
- `-> ioWriter`  
The PDI writer object, which was created by a previous call to the PdiWriterNew function.
- `-> charset`  
The character set to use for the property value. This must be one of the following CharEncodingType values:
  - charEncodingAscii
  - charEncodingISO8859_1
  - charEncodingShiftJIS
  - charEncodingISO2022Jp

**Result**  
Returns errNone if successful, and an error code if not successful.

**Comments**  
This function tells ioWriter to use the specified charset for the next property value that it writes, rather than computing a character set for that value.

You can determine the current character setting by examining the charset field of your PDI writer object.

**Compatibility**  
Implemented only if 4.0 New Feature Set is present.

**See Also**  
PdiSetEncoding
PdiSetEncoding

**Purpose**
Force the encoding of the current property value.

**Declared In**
PdiLib.h

**Prototype**
```
Err PdiSetEncoding(UInt16 libRefnum,
PdiWriterType *ioWriter, UInt16 encoding)
```

**Parameters**
- `libRefnum`  
The PDI library reference number.
- `ioReader`  
The PDI writer object, which was created by a previous call to the PdiWriterNew function.
- `encoding`  
The encoding to apply to the property value. This must be one of the Encoding Type Constants.

**Result**
Returns `errNone` if successful, and an error code if not successful.

**Comments**
This function changes the encoding for the property value to the specified encoding value.
You can determine the current encoding setting by examining the encoding field of your PDI writer object.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
PdiSetCharset

PdiWriteBeginObject

**Purpose**
Writes a vObject begin tag to an output stream.

**Declared In**
PdiLib.h

**Prototype**
```
Err PdiWriteBeginObject(UInt16 libRefnum,
PdiWriterType *ioWriter, UInt16 objectNameID)
```

**Parameters**
- `libRefnum`  
The PDI library reference number.
-> ioWriter
   The PDI writer object, which was created by a previous call to the PdiWriterNew function.

-> objectNameID
   The object name ID. This must be one of the Property Name Constants that begins an object, including the following:
   kPdiPRN_BEGIN_VCAL
   kPdiPRN_BEGIN_VCAL
   kPdiPRN_BEGIN_VCARD
   kPdiPRN_BEGIN_VEVENT
   kPdiPRN_BEGIN_VFREEBUSY
   kPdiPRN_BEGIN_VJOURNAL
   kPdiPRN_BEGIN_VTIMEZONE
   kPdiPRN_BEGIN_VTODO

Result
   Returns errNone if successful, and an error code if not successful.

Comments
   Call this function to begin writing a vObject to the output stream. It writes a begin tag such as “BEGIN:VCARD” to the output stream.

Compatibility
   Implemented only if 4.0 New Feature Set is present.

See Also
   PdiWriteEndObject, PdiWriteProperty

PdiWriteEndObject

Purpose
   Writes a vObject end tag to an output stream.

Declared In
   PdiLib.h

Prototype
   Err PdiWriteEndObject(UInt16 libRefnum, PdiWriterType *ioWriter, UInt16 objectNameID)

Parameters
   -> libRefnum
   The PDI library reference number.
-> ioWriter  The PDI writer object, which was created by a previous call to the PdiWriterNew function.

-> objectNameID  The object name ID. This must be one of the Property Name Constants that ends an object, including the following:

kPdiPRN_END_VCAL
kPdiPRN_END_VCAL
kPdiPRN_END_VCARD
kPdiPRN_END_VEVENT
kPdiPRN_END_VFREEBUSY
kPdiPRN_END_VJOURNAL
kPdiPRN_END_VTIMEZONE
kPdiPRN_END_VTODO

Result  Returns errNone if successful, and an error code if not successful.

Comments  Call this function to finish writing a vObject to the output stream. It writes a end tag such as “END:VCARD” to the output stream.

Compatibility  Implemented only if 4.0 New Feature Set is present.

See Also  PdiWriteBeginObject, PdiWriteProperty

PdiWriteParameter

Purpose  Write a parameter, and optionally its name, to an output stream.

Declared In  PdiLib.h

Prototype  Err PdiWriteParameter(UInt16 libRefnum, PdiWriterType *ioWriter, UInt16 parameter, Boolean parameterName)

Parameters  -> libRefnum  The PDI library reference number.
-> ioWriter  The PDI writer object, which was created by a previous call to the PdiWriterNew function.

-> parameter The ID of the parameter. This must be one of the Parameter Value Constants.

-> parameterName
  If this is true, the parameter name, followed by the “=” symbol, followed by the parameter value is written to the output stream.
  If this is false, only the parameter value is written to the output stream.

Result  Returns errNone if successful, and an error code if not successful.

Comments  Use this function to write a parameter to the output stream. To write a property, you usually call the PdiWriteProperty function, followed by calls to PdiWriteParameter to write any parameters, followed by a call to the PdiWritePropertyValue function to write the property value. For more information, see Writing Properties in Chapter 3, “Personal Data Interchange,” in Palm OS Programmer’s Companion, vol. II, Communications.

You can use the parameterName argument to specify that you want the parameter name written as well as the parameter value. For example, the following table shows what is written if the value of parameter is kPdiPAV_TYPE_HOME.

<table>
<thead>
<tr>
<th>Value of parameterName</th>
<th>Data written to output stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>TYPE=HOME</td>
</tr>
<tr>
<td>false</td>
<td>HOME</td>
</tr>
</tbody>
</table>

Compatibility  Implemented only if 4.0 New Feature Set is present.

See Also  PdiWriteProperty, PdiWritePropertyValue, PdiWritePropertyFields, PdiWritePropertyStr, PdiWriteParameterStr
PdiWriteParameterStr

**Purpose**
Write a parameter name and the parameter value to an output stream.

**Declared In**
PdiLib.h

**Prototype**
Err PdiWriteParameterStr(UInt16 libRefnum, PdiWriterType *ioWriter, const Char *parameterName, const Char *parameterValue)

**Parameters**
- **libRefnum**
The PDI library reference number.
- **ioWriter**
The PDI writer object, which was created by a previous call to the PdiWriterNew function.
- **parameterName**
The name of the parameter. If the value of this is the empty string or NULL, only the parameter value is written.
- **parameterValue**
The parameter value string.

**Result**
Returns errNone if successful, and an error code if not successful.

**Comments**
This function writes the parameter name, followed by the “=” symbol, followed by the parameter value, to the output stream. If parameterName is NULL, or if its value is the empty string, just the parameter value is written.

This function is similar to the PdiWriteParameter function. The difference is that PdiWriteParameterStr takes the name and value of the parameter as strings, while PdiWriteParameter takes them as ID constants.
Compatibility
 Implemented only if 4.0 New Feature Set is present.

See Also
PdiWriteProperty, PdiWritePropertyValue, PdiWritePropertyFields, PdiWritePropertyStr, PdiWriteParameter

PdiWriteProperty

Purpose
 W rites a property name to an output stream.

Declared In
 PdiLib.h

Prototype
Err PdiWriteProperty(UInt16 libRefnum, PdiWriterType *ioWriter, UInt16 propertyNameID)

Parameters
- > libRefnum The PDI library reference number.
- > ioWriter The PDI writer object, which was created by a previous call to the PdiWriterNew function.
- > propertyNameID The property name to write. This must be one of the Property Name Constants.

Result
 Returns errNone if successful, and an error code if not successful.

Comments
 To write a property, you usually call PdiWriteProperty, followed by calls to the PdiWriteParameter function to write any parameters, followed by a call to the PdiWritePropertyValue function to write the property value. For more information, see Writing Properties in Chapter 3, “Personal Data Interchange,” in Palm OS Programmer’s Companion, vol. II, Communications.

Compatibility
 Implemented only if 4.0 New Feature Set is present.

See Also
PdiWritePropertyValue, PdiWritePropertyFields, PdiWritePropertyStr, PdiWriteParameter
**PdiWritePropertyBinaryValue**

**Purpose**
Write a binary property value to an output stream.

**Declared In**
PdiLib.h

**Prototype**
```c
Err PdiWritePropertyBinaryValue(UInt16 libRefnum, PdiWriterType *ioWriter, const Char *buffer, UInt16 size, UInt16 options)
```

**Parameters**
- `libRefnum` The PDI library reference number.
- `ioWriter` The PDI writer object, which was created by a previous call to the `PdiWriterNew` function.
- `buffer` A buffer that contains the binary data.
- `size` The number of bytes of data to write from the buffer.
- `options` The data type. This must be a combination of one or more of the Value Type Constants.

**Result**
Returns erNone if successful, and an error code if not successful.

**Comments**
Use this function to write a binary data property value, such as a sound or an image.

This function encodes the data when it is written. The character set that gets applied to the data is not computed by this function; however, you can call the `PdiSetCharset` function to set the character set.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
PdiWriteProperty, PdiWritePropertyFields, PdiWritePropertyValue
PdiWritePropertyFields

Purpose
Write a structured property value with multiple fields to an output stream.

Declared In
PdiLib.h

Prototype
Err PdiWritePropertyFields(UInt16 libRefnum, PdiWriterType *ioWriter, Char *fields[], UInt16 fieldNumber, UInt16 options)

Parameters
- libRefnum The PDI library reference number.
- ioWriter The PDI writer object, which was created by a previous call to the PdiWriterNew function.
- fields An array of strings, each of which is a field of the property value. Individual fields can be NULL.
- fieldNumber The number of fields in the Fields array.
- options The data type. This must be a combination of one or more of the Value Type Constants.

Result
Returns errNone if successful, and an error code if not successful.

Comments
Use this function to write a property value that contains multiple fields.

Compatibility
Implemented only if 4.0 New Feature Set is present.

See Also
PdiWritePropertyValue, PdiWritePropertyBinaryValue, PdiReadPropertyField
PdiWritePropertyStr

**Purpose**
Writes the name of a property to the output stream, and specifies the property value’s structure for subsequent write operations.

**Declared In**
PdiLib.h

**Prototype**
Err PdiWritePropertyStr(UInt16 libRefnum, PdiWriterType *ioWriter, const Char *propertyName, UInt8 writeMode, UInt8 requiredFields)

**Parameters**
- -> libRefnum The PDI library reference number.
- -> ioWriter The PDI writer object, which was created by a previous call to the PdiWriterNew function.
- -> propertyName The name of the property to write.
- -> writeMode The format of the fields in the property value. Use one of the following Property Value Format Constants:
  - kPdiNoFields
  - kPdiCommaFields
  - kPdiSemicolonFields
- -> requiredFields The number of required fields for the value. This is usually set to 1.

**Result**
Returns errNone if successful, and an error code if not successful.

**Comments**
Use this function when you are writing a property that is not in the dictionary, or when you are writing a property that uses value formatting that differs from the standard formatting stored in the dictionary for the property name.

This function writes the property name to the output stream, and then establishes the structure of the property’s value, including the number of required fields and the separator character for those
fields. After calling this function, the next call to the
PdiWritePropertyValue or PdiWritePropertyFields
functions correctly writes the property value.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also PdiWriteProperty, PdiWritePropertyValue,
PdiWritePropertyFields, PdiWriteParameter

PdiWritePropertyValue

Purpose Write a string to the output stream as the entire value of a property.

Declared In PdiLib.h

Prototype Err PdiWritePropertyValue(UInt16 libRefnum,
PdiWriterType *ioWriter, Char *buffer,
UInt16 options)

Parameters
- > libRefnum The PDI library reference number.
- > ioWriter The PDI writer object, which was created by a
previous call to the PdiWriterNew function.
- > buffer The input buffer that contains the string to be
written.
- > options The data type. This must be a combination of
one or more of the Value Type Constants.

Result Returns errNone if successful, and an error code if not successful.

Comments Use this function to write a property value that contains a single
field.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also PdiWriteProperty, PdiWritePropertyFields,
PdiWriteParameter, PdiWritePropertyBinaryValue
PdiWriterDelete

**Purpose**
Delete a PDI output stream object.

**Declared In**
PdiLib.h

**Prototype**
```c
void PdiWriterDelete(UInt16 libRefnum, PdiWriterType **ioWriter)
```

**Parameters**
- `libRefnum` The PDI library reference number.
- `ioWriter` A pointer to the PDI writer object, which was created by a previous call to the PdiWriterNew function.

**Result**
Returns nothing.

**Comments**
This function frees the memory that was allocated for the writer object. The `ioWriter` parameter is set to NULL.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
PdiWriterNew

PdiWriterNew

**Purpose**
Initializes a new PDI writer object for use with the specified library number.

**Declared In**
PdiLib.h

**Prototype**
```c
PdiWriterType *PdiWriterNew(UInt16 libRefnum, UDAWriterType *output, UInt16 version)
```

**Parameters**
- `libRefnum` The PDI library reference number.
- `output` The Unified Data Access (UDA) output object associated with the writer.
- version  Options to control the behavior of the writer. You can use a combination of the Reader and Writer Options Constants.

**Result**  Returns a pointer to the new PDI writer object. Returns **NULL** if the reader cannot be created.

**Comments**  The `media` pointer is copied into a field in the writer object; thus, you do not need to retain your copy.

**Compatibility**  Implemented only if 4.0 New Feature Set is present.

**See Also**  [PdiWriterDelete](#), [PdiReaderNew](#)
Unified Data Access Manager

This chapter provides reference material for the Unified Data Access (UDA) Manager, which provides a mechanism for abstracting read and write access to different kinds of source and destination media, including memory and the Exchange Manager.

The Personal Data Interchange (PDI) reader and writer objects use UDA objects, and you must create UDA objects to use the PDI functions.

This chapter discusses the following topics:

- **UDA Manager Data Structures**
- **UDA Manager Constants**
- **UDA Manager Functions**
- **UDA Object Creation Functions**

The header file `UDAMgr.h` declares the Unified Data Access Manager API.

You use the UDA Manager in conjunction with the PDI library. For more information about the PDI library, see Chapter 85, “Personal Data Interchange Library.”

Chapter 3, “Personal Data Interchange,” in the Palm OS Programmer’s Companion, vol. II, Communications, provides examples of using the UDA functions with the PDI library.

UDA Manager Data Structures

**UDABufferSize**

The `UDABufferSize` type is a simple typedef that defines the size of buffers used with UDA read functions.
typedef Uint16 UDABufferSize

UDAObjectType

The UDAObjectType is the base class for all UDA objects, and defines the common properties of all of the objects.

typedef struct UDAObjectTag {
    Uint16               optionFlags;
    UDADeleteFunction    deleteF;
    UDAControlFunction   controlF;
} UDAObjectType

Field Descriptions

optionFlags       Options for the object. This is a combination of values described in Object Option Flags.
deleteF           The delete function associated with this UDA object.
controlF          The control function associated with this UDA object.

UDAFilterType

The UDAFilterType represents UDA Filters, which take input from a UDA Reader or UDA Filter, perform some encoding or decoding operation, and output the data to a memory buffer.

typedef struct UDAFilterTag {
    Uint16               optionFlags;
    UDADeleteFunction    deleteF;
    UDAControlFunction   controlF;
    UDAReadFunction      readF;
    UDAReaderType*       upperReader;
} UDAFilterType
Field Descriptions

optionFlags
Options for the object. This is a combination of values described in Object Option Flags.

deleteF
The delete function associated with this UDA object.

controlF
The control function associated with this UDA object.

readF
The read function associated with this UDA object.

upperReader
The UDAReaderType or UDAFillterType object that reads the data that this object outputs.

UDAReaderType

The UDAReaderType represents UDA Readers, which read input from a medium.

typedef struct UDAReaderTag {
    UInt16 optionFlags;
    UDADeleteFunction deleteF;
    UDAControlFunction controlF;
    UDAReadFunction readF;
} UDAReaderType

Field Descriptions

optionFlags
Options for the object. This is a combination of values described in Object Option Flags.

deleteF
The delete function associated with this UDA object.
UDAWriterType

The `UDAWriterType` represents UDA Writers, which take data from a UDA Reader or UDA Filter and write the data to an output medium.

```c
typedef struct UDAWriterTag {
    UInt16               optionFlags;
    UDADeleteFunction    deleteF;
    UDAControlFunction   controlF;
    UDAWriteFunction     initiateWriteF;
    UDADeleteFunction    controlF;
    UDADeleteFunction    deleteF;
    UDAFlushFunction     flushF;
    UDAReaderType*       upperReader;
} UDAWriterType
```

**Field Descriptions**

- `optionFlags`: Options for the object. This is a combination of values described in [Object Option Flags](#).
- `deleteF`: The delete function associated with this UDA object.
- `controlF`: The control function associated with this UDA object.
- `initiateWriteF`: The write function associated with this UDA object.
- `flushF`: The flush function associated with this UDA object.
- `upperReader`: The `UDAReaderType` object that reads the data that this object writes.
UDA Manager Constants

This section describes the constants used with the UDA Manager, which include the following constant types:

- Control Flags
- Error Constants
- Object Option Flags
- Miscellaneous Constants

Control Flags

Use the control flag constants to control UDA objects with the UDAControl macro.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kUDAReinitialize</td>
<td>1</td>
<td>Used with the UDAControl macro to reinitialize the UDA object.</td>
</tr>
</tbody>
</table>

Error Constants

At the time of this writing, there is only one error constant associated with the UDA object API.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>udaErrControl</td>
<td>Returned by the UDAControl macro when the control parameter is not valid for the UDA object.</td>
</tr>
</tbody>
</table>

Object Option Flags

You use the object option flag constants to determine information about the internal state of UDA objects. Note that the UDAEndOfReader and UDAMoreData macros provide you with a convenient means of accessing this same information.
UDA Manager Functions

UDAControl

**Purpose**
Applies controls to a UDA object.

**Declared In**
UDAMgr.h

**Prototype**
Err UDAControl (UDAObjectType* ioObjectP, UInt16 parameter, va_args)

**Parameters**
- `-> ioObjectP` A pointer to the UDAOBJECT object that you want to control. This can be a UDAREADERTYPE, a UDAFILTERTYPE, or a UDAWRITERTYPE object.
- `-> parameter` The control action that you want applied to the object.

---

### Miscellaneous Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kUDAEndOfReader</td>
<td>1</td>
<td>Indicates that the UDA reader has reached the end of its data.</td>
</tr>
<tr>
<td>kUDAMoreData</td>
<td>2</td>
<td>Indicates that the UDA reader needs more space in the read buffer to do its work.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kUDAZeroTerminatedBuffer</td>
<td>0xFFFF</td>
<td>Indicates that the buffer is a null-terminated string. Use this value when creating or reinitializing a UDAMemoryReader object.</td>
</tr>
</tbody>
</table>
-> va_args Additional parameters, as required for the control and object type.

**Result**
Returns `errNone` if no error, or `udaErrorClass` if the control parameter is not valid for the `ioObjectP`.

**Comments**
The `UDAControl` function applies a control action to a UDA object. You may need to supply additional parameters, depending on the object type and control parameter values.

The only control action defined in Palm OS 4.0 is `kUDAReinitialize`. You can use it as shown in Table 86.1.

**Table 86.1 UDAControl actions**

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Usage</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDAExchangeReaderType</td>
<td><code>UDAControl(myExgRdr, kUDAReinitialize)</code></td>
<td>Does nothing</td>
</tr>
<tr>
<td>UDAExchangeWriterType</td>
<td><code>UDAControl(myExgWtr, kUDAReinitialize)</code></td>
<td>Does nothing</td>
</tr>
<tr>
<td>UDAMemoryReaderType</td>
<td><code>UDAControl(myMemRdr, kUDAReinitialize, bufferP, bufferSize)</code></td>
<td>Reinstalls a new buffer for the memory reader. See <a href="#">UDAMemoryReaderNew</a> for more information about the parameters.</td>
</tr>
</tbody>
</table>

**Compatibility**
Implemented only if [4.0 New Feature Set](#) is present.
**UDADelete**

**Purpose**
Macro that deletes a UDA object.

**Declared In**
UDAMgr.h

**Prototype**
UDADelete (ioObjectP)

**Parameters**
- `-> ioObjectP` A pointer to the UDAObjectType object that you want to delete. This can be a UDAReaderType, a UDAPFilterType, or a UDAWriterType object.

**Result**
Returns nothing.

**Comments**
The ioObjectP pointer is not valid after this macro completes.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**UDAEndOfReader**

**Purpose**
Macro that tests if the end of the reader has been reached.

**Declared In**
UDAMgr.h

**Prototype**
UDAEndOfReader (ioReaderP)

**Parameters**
- `-> ioReaderP` A pointer to a UDAReaderType object.

**Result**
Returns true if the end of the reader referenced by ioReaderP has been reached, and false if not.

**Comments**
The end of the reader has been reached.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.
UDAFilterJoin

Purpose  Macro that joins a filter with a reader.

Declared In  UDAMgr.h

Prototype  UDAFilterJoin (ioFilterP, newReaderP)

Parameters
  --> ioFilterP  A pointer to a UDAFilterType object.
  --> newReaderP  A pointer to the UDAReaderType object with which you want the filter joined.

Result  Returns nothing.

Comments  Each UDAFilterType object receives its data from the UDAReaderType object to which it is joined; this relationship is established when you create the filter object. You can use this macro to change the reader with which the filter is joined. Upon completion, the filter referenced by ioFilterP is joined with the reader referenced by newReaderP.

Compatibility  Implemented only if 4.0 New Feature Set is present.

UDAInitiateWrite

Purpose  Macro that causes the UDAWriterType object to read data and then write that data to output.

Declared In  UDAMgr.h

Prototype  UDAInitiateWrite (ioWriterP)

Parameters  --> ioWriterP  A pointer to a UDAWriterType object.

Result  Returns errNone if successful, and an error code if not.
Comments When you use this macro, the \texttt{ioWriterP} reads data from the reader to which it is joined. It reads data until the reader is empty, and then writes the data to the output medium.

Compatibility Implemented only if \textit{4.0 New Feature Set} is present.

**UDAMoreData**

Purpose Macro that tests if there is more data available to read, but not enough room in the buffer to read it in.

Declared In \texttt{UDAMgr.h}

Prototype \texttt{UDAMoreData} (\texttt{ioReaderP})

Parameters \- \texttt{ioReaderP} A pointer to a \texttt{UDAREaderType} object.

Result Returns \texttt{true} if there is more data available for the reader and \texttt{false} if there is no more data available.

Comments You can use this macro with \texttt{UDAREaderType} objects to determine if there is more data waiting to read. This can happen when the reader’s buffer is full.

Compatibility Implemented only if \textit{4.0 New Feature Set} is present.

**UDARead**

Purpose Macro that uses the specified \texttt{UDAREaderType} object to read data from the input source and place that data into the specified buffer.

Declared In \texttt{UDAMgr.h}

Prototype \texttt{UDARead} (\texttt{ioReaderP}, \texttt{bufferToFillP}, \texttt{bufferSize}, \texttt{errorP})

Parameters \- \texttt{ioReaderP} A pointer to a \texttt{UDAREaderType} object that performs the read.
-> bufferToFillP  A pointer to the buffer into which data is placed.

-> bufferSize  The size of the buffer, in bytes.

-> errorP  A pointer to an Err value that represents the result of the read operation; if the operation is successful, the value is set to errNone.

**Result**  Returns the number of bytes that were actually read. This value can be less than or equal to the value of bufferSize.

**Comments**  The reader reads from the input source associated with the reader object and places the data into the specified buffer, reading no more than bufferSize bytes of data.

**Compatibility**  Implemented only if 4.0 New Feature Set is present.

---

**UDAWriterFlush**

**Purpose**  Macro that flushes the buffer used by the UDAWriterType object.

**Declared In**  UDAMgr.h

**Prototype**  UDAWriterFlush (ioWriterP)

**Parameters**

-> ioWriterP  A pointer to a _UDAWriterType object.

**Result**  Returns errNone if successful, and an error code if not.

**Comments**  You can use this macro to flush any data remaining in the buffer of the writer object referenced by ioWriterP. This causes any data in the buffer to be sent to the output medium.

**Compatibility**  Implemented only if 4.0 New Feature Set is present.
UDAWriterJoin

**Purpose**
Macro that joins a writer object to a different reader object.

**Declared In**
UDAMgr.h

**Prototype**
UDAWriterJoin (ioWriterP, newReaderP)

**Parameters**
- `ioWriterP` A pointer to a UDAWriterType object.
- `newReaderP` A pointer to the UDAReaderType object with which you want the writer joined.

**Result**
Returns nothing.

**Comments**
Each UDAWriterType object receives its data from the UDAReaderType object to which it is joined; this relationship is established when you create the writer object. You can use this macro to change the reader with which the writer is joined. Upon completion, the writer referenced by `ioWriterP` is joined with the reader referenced by `newReaderP`.

**Compatibility**
Implemented only if 4.0 Feature Set is present.
UDA Object Creation Functions

**UDAExchangeReaderNew**

**Purpose**
Creates a new `UDAReaderType` object that you can use to read data from an Exchange Manager socket.

**Declared In**
UDAMgr.h

**Prototype**

```c
UDAReaderType* UDAExchangeReaderNew (ExgSocketPtr socket)
```

**Parameters**

- `socket` A pointer to an `ExgSocketType` structure that describes the connection.

**Result**
Returns a pointer to the newly created `UDAReaderType` object, or `NULL` if the reader could not be created.

**Comments**
Use this function to create a UDA Reader object that takes input from an Exchange Manager socket.

**Compatibility**
Implemented only if 4.0 New Feature Set is present.

**See Also**
`ExgSocketType`

**UDAExchangeWriterNew**

**Purpose**
Creates a new `UDAWriterType` object that you can use to write data to an Exchange Manager socket.

**Declared In**
UDAMgr.h

**Prototype**

```c
UDAWriterType* UDAExchangeWriterNew (ExgSocketPtr socket, UDABufferSize bufferSize)
```

**Parameters**

- `socket` A pointer to an `ExgSocketType` structure that describes the connection.
-> bufferSize   The size, in bytes, of the buffer for the new writer object.

**Result**   Returns a pointer to the newly created UDA Writer, or NULL if the writer could not be created.

**Comments**   Use this function to create a UDA Writer object that sends output to an Exchange Manager socket.

**Compatibility**   Implemented only if *4.0 New Feature Set* is present.

**See Also**   ExgSocketType

---

**UDAMemoryReaderNew**

**Purpose**   Creates a new UDAMemoryReader object that you can use to read data from a memory buffer.

**Declared In**   UDAMgr.h

**Prototype**   UDAMemoryReader* UDAMemoryReaderNew  
(const UInt8* bufferP,  
UDABufferSize bufferSizeInBytes)

**Parameters**
- -> bufferP   A pointer to a buffer in memory from which data is read.
- -> bufferSize   The size of the buffer, in bytes. If this value is equal to kUDAZeroTerminatedBuffer, bufferP must point to a null-terminated string buffer.

**Result**   Returns a pointer to the newly created UDAMemoryReader object, or NULL if the reader could not be created.

**Comments**   Use this function to create a reader that takes input from memory.
**Compatibility**  Implemented only if 4.0 New Feature Set is present.
Part V: Appendixes
System Use Only Functions

This appendix lists functions that are purposely undocumented because they are for system use only.

**WARNING!** System Use Only.

- AbtShowAbout
- AlmAlarmCallback
- AlmCancelAll
- AlmDisplayAlarm
- AlmEnableNotification
- AlmInit
- AlmTimeChange
- BtLibHandleEvent
- BtLibHandleTransportEvent
- BtLibMutex
- BtLibOpenBackground
- BtLibServiceClose
- BtLibServiceIndicateSessionStart
- BtLibServiceOpen
- BtLibServicePlaySound
- BtLibSleep
- BtLibUnload
- BtLibWake
- Crc16CalcBigBlock
- DmInit
- DmResetRecordStates
- ErrAlertCustom
- EvtDequeueKeyEvent
- EvtEnqueuePenPoint
- EvtGetSysEvent
- EvtInitialize
System Use Only Functions

EvtSetKeyQueuePtr
EvtSetPenQueuePtr
EvtSysInit
ExgInit
ExgNotifyReceiveV35
ExpCardInserted
ExpCardRemoved
ExpInit
ExpSlotRegister
ExpSlotUnregister
FileReadLow
Find
FrmActiveState
FrmAddSpaceForObject
FrmGetUserModifiedState
FrmSetNotUserModified
FtrInit
GrfFieldChange
GrfFree
GrfInit
HostControl
INetLibSleep
INetLibSockMailAttrGet
INetLibSockMailAttrSet
INetLibSockMailQueryProgress
INetLibSockMailReqAdd
INetLibSockMailReqCreate
INetLibSockMailReqSend
INetLibWake
InsPtCheckBlink
InsPtInitialize
IntlInit
IrHandleEvent
IrWaitForEvent
KeyboardStatusNew
KeyboardStatusFree
KbdSetLayout
KbdGetLayout
KbdSetPosition
KbdGetPosition
System Use Only Functions

KbdSetShiftState  
KbdGetShiftState  
KbdDraw  
KbdErase  
KbdHandleEvent  
MemCardFormat  
MemChunkFree  
MemChunkNew  
MemHandleFlags  
MemHandleLockCount  
MemHandleOwner  
MemHandleResetLock  
MemHeapFreeByOwnerID  
MemHeapInit  
MemInit  
MemInitHeapTable  
MemKernelInit  
MemPtrFlags  
MemPtrOwner  
MemPtrResetLock  
MemSemaphoreRelease  
MemSemaphoreReserve  
MemStoreSetInfo  
NetLibConfigAliasGet  
NetLibConfigAliasSet  
NetLibConfigDelete  
NetLibConfigIndexFromName  
NetLibConfigList  
NetLibConfigMakeActive  
NetLibConfigRename  
NetLibConfigSaveAs  
NetLibHandlePowerOff  
NetLibOpenConfig  
NetLibOpenIfCloseWait  
NetLibSleep  
NetLibWake  
PenClose  
PenGetRawPen  
PenOpen  
PenRawToScreen
System Use Only Functions

PenScreenToRaw
PenSleep
PenWake
ResLoadString
ScrCompressScanLine
ScrCopyRectangle
ScrDeCompressScanLine
ScrDrawChars
ScrDrawNotify
ScrInit
ScrLineRoutine
ScrRectangleRoutine
ScrScreenInfo
ScrSendUpdateArea
SerDbgAssureOpen
SerialMgrInstall
SerReceiveISP
SerReceiveWindowClose
SerReceiveWindowOpen
SerSetMapPort
SerSetWakeupHandler
SerSleep
SerWake
SlkProcessRPC
SlkSysPktDefaultResponse
SndInit
SndSetDefaultVolume
SrmSelectorErrPrv
SrmSleep
SrmWake
SysAppStartup
SysAppExit
SysBatteryDialog
SysCardImageDeleted
SysCardImageInfo
SysColdBoot
SysDisableInts
SysDoze
SysEvGroupCreate
SysEvGroupRead
System Use Only Functions

SysEvGroupSignal
SysEvGroupWait
SysInit
SysKernelInfo
SysLaunchConsole
SysLCDBrightness
SysLCDContrast
SysLibClose
SysLibInstall
SysLibOpen
SysLibSleep
SysLibTblEntry
SysLibWake
SysMailboxCreate
SysMailboxDelete
SysMailboxFlush
SysMailboxSend
SysMailboxWait
SysNewOwnerID
SysPowerOn
SysResSemaphoreCreate
SysResSemaphoreDelete
SysResSemaphoreRelease
SysResSemaphoreReserve
SysRestoreStatus
SysSemaphoreCreate
SysSemaphoreDelete
SysSemaphoreSet
SysSemaphoreSignal
SysSemaphoreWait
SysSetA5
SysSetPerformance
SysSleep
SysTaskCreate
SysTaskDelete
SysTaskID
SysTaskResume
SysTaskSetTermProc
SysTaskSuspend
SysTaskSwitching
System Use Only Functions

SysTaskTrigger
SysTaskWait
SysTaskWaitClr
SysTaskWake
SysTimerCreate
SysTimerDelete
SysTimerRead
SysTimerWrite
SysTranslateKernelErr
Sys UIBusy
SysUILaunch
SysUnimplemented
SysWantEvent
TimInit
TxtPrepFindString
UIColorPopTable
UIColorPushTable
UINitialize
UIReset
UIPopTable
UIPushTable
VFSInit
WinAddWindow
WinDrawWindowFrame
WinDisableWindow
WinEnableWindow
WinGetWindowPointer
WinInitializeWindow
WinMoveWindowAddr
WinRemoveWindow
WinScreenInit
WinSetColors
Compatibility Guide

This appendix lists groups of functions and other features (such as events and launch codes) that have been added to the Palm OS® after version 1.0.

Before you use any new functions or features in an application, you must check to ensure that they are implemented in the OS version your application is running on. Checking the OS version number is not a reliable indicator that a specific feature is present, since some later OS versions do not include features present in earlier versions. In order to ensure that your code is supported, you must check for the presence of individual features.

To make this check easier, this appendix lists new functions and features in groups such that all functions and features in a group are always implemented together in the ROM of a Palm™ device. This means that you can check for a single feature in that group and be assured that if that feature is present than all functions and features in that group are implemented.

Each group includes a recommended test to check if it is implemented. The following groups are described:

- 2.0 New Feature Set
- 3.0 New Feature Set
- 3.1 New Feature Set
- 3.2 New Feature Set
- International Feature Set
- Japanese Feature Set
- Wireless Internet Feature Set
- New Serial Manager Feature Set
- Connection Manager Feature Set
- 3.5 New Feature Set
- Notification Feature Set
2.0 New Feature Set

You can check that this feature set is implemented by checking that the system version is 2.0 or higher. Use this FtrGet call:

```c
err = FtrGet(sysFtrCreator,
    sysFtrNumROMVersion, &romversion);
```

The `romversion` parameter should be `0x02003000` or greater.

Launch Codes

This feature set adds the following launch codes:

```c
sysAppLaunchCmdLookup
sysAppLaunchCmdPanelCalledFromApp
sysAppLaunchCmdReturnFromPanel
sysAppLaunchCmdSystemLock
```

Functions

This feature set adds the following functions:

```c
CategoryInitialize
DmDeleteCategory
EvtAddUniqueEventToQueue
EvtSysEventAvail
```

```c
CategorySetName
DmDatabaseProtect
EvtEventAvail
```
Existing Functions that Changed

Several functions that existed in 1.0 were changed in 2.0:

- FldGetNumberOfBlankLines
- FldSetInsertionPoint
- FntGetScrollValues
- FntWordWrap
- FntWordWrapReverseNLines
- FrmPointInTitle
- FrmSetObjectBounds
- KeySetMask
- LocGetNumberSeparator
- LstScrollList
- LstGetVisibleItems
- MemCmp
- MenuSetActiveMenuRscID
- PhoneNumberLookup
- PrefSetPreference
- PrefGetPreference
- SclDrawScrollBar
- SclHandleEvent
- SclGetScrollBar
- SclSetScrollBar
- SerControl
- StrDelocalizeNumber
- StrNCaselessCompare
- StrNCompare
- StrPrintF
- SysBinarySearch
- SysCreatePanelList
- SysGraffitiReferenceDialog
- SysStringByIndex
- SysCreateDataBaseList
- SysErrString
- SysLibLoad
- SysTicksPerSecond
- TblHasScrollBar
- TblSetColumnEditIndicator
- TblSetBounds
- TblSetRowStaticHeight
- WinSetWindowBounds
Other Changes

As a rule, all Palm OS applications developed with the 1.0 SDK should run error-free on the latest device. There are two possible pitfalls for 1.0 applications:

- **fldChangedEvent Change**—The operating system now correctly sends a fldChangedEvent whenever a field object is changed. Previously, the event was at times not sent, especially when a FldSetText operation was performed. If your application doesn’t catch the events that are now sent, it may have problems.

- **Non-standard tools**—If your application was not developed with Metrowerks Code Warrior for the Palm OS, it may run
into problems. One known problem can occur if the application:

- was compiled with optimization turned on
- uses system preferences
3.0 New Feature Set

You can check that this feature set is implemented by checking that the system version is 3.0 or higher. Use this `FtrGet` call:

```c
err = FtrGet(sysFtrCreator,
               sysFtrNumROMVersion, &romVersion);
```

The `romVersion` parameter should be greater than or equal to `0x03003000`, which can be constructed using the `sysMakeROMVersion` macro:

```c
sysMakeROMVersion(3,0,0,sysROMStageRelease,0)
```

Launch Codes

This feature set adds the following launch codes:

- `sysAppLaunchCmdExgAskUser`
- `sysAppLaunchCmdExgReceiveData`

In addition, the launch code `sysAppLaunchCmdGoto` is now also sent by the exchange manager, in addition to its use by the global find operation.

Font

This feature set adds the following font:

`largeBoldFont`

Functions

This feature set adds the following functions:

Dynamic User Interface Functions

- `CtlNewControl`
- `CtlValidatePointer`
- `FldNewField`
- `FrmNewBitmap`
- `FrmNewForm`
- `FrmNewGadget`
- `FrmNewLabel`
- `FrmRemoveObject`
- `FrmValidatePtr`
- `LstNewList`
- `WinValidateHandle`
For more information on creating and using dynamic user interface elements, see the section “Dynamic UI” on page 142 of the Palm OS Programmer’s Companion, vol. I.

**Font Functions**

FontSelect  
FntDefineFont

For more information on these functions and the support for custom fonts, see “Fonts” on page 268 of the Palm OS Programmer’s Companion, vol. I.

**Progress Manager Functions**

PrgHandleEvent  
PrgStartDialog  
PrgStopDialog  
PrgUpdateDialog  
PrgUserCancel

For more information, see the section “Progress Dialogs” on page 86 of the Palm OS Programmer’s Companion, vol. I.

**File Streaming Functions**

FileClearerr  
FileClose  
FileControl  
FileDelete  
FileDmRead  
FileEOF  
FileError  
FileFlush  
FileGetLastError  
FileOpen  
FileRead  
FileReadLow (system use only)  
FileRewind  
FileSeek  
FileTell  
FileTruncate  
FileWrite

For more information, see the section “File Streaming Application Program Interface” on page 202 of the Palm OS Programmer’s Companion, vol. I.

**Sound Functions**

SndCreateMidiList  
SndPlaySmf  
SndDoCmd (enhanced in 3.0)
Exchange Manager Functions

ExgAccept    ExgPut
ExgDBRead    ExgReceive
ExgDBWrite   ExgRegisterData
ExgDisconnect ExgSend

For more information, see the chapter Beaming (Infrared Communication) in the Palm OS Programmer’s Companion, vol. II, Communications.

IR Library Functions

IrAdvanceCredit    IrIAS_GetUserStringLen
IrBind
IrClose
IrConnectIrLap
IrConnectReq
IrConnectRsp
IrDataReq
IrDisconnectIrLap
IrDiscoverReq
IrIAS_Add
IrIAS_GetInteger
IrIAS_GetIntLsap
IrIAS_GetObjectID
IrIAS_GetOctetString
IrIAS_GetOctetStringLength
IrIAS_GetType
IrIAS_GetUserString
IrIAS_GetUserStringCharSet
IrUnbind
IrIsIrLapConnected
IrIsMediaBusy
IrIsNoProgress
IrIsRemoteBusy
IrLocalBusy
IrMaxRxSize
IrMaxTxSize
IrOpen
IrSetConTypeLMP
IrSetConTypeTTP
IrSetDeviceInfo
IrTestReq

For more information, see the chapter Beaming (Infrared Communication) in the Palm OS Programmer’s Companion, vol. II, Communications.
Miscellaneous Functions

- `FrmRestoreActiveState`
- `FrmSaveActiveState`
- `ScrDisplayMode`
- `SysGetAppInfo (system use only)`
- `SysGetOSVersionString`

Existing Functions that Changed

Two functions that existed in 2.0 were changed in 3.0:

- `CategoryEdit` (old function renamed `CategoryEditV20`)
- `SysBatteryInfo` (old function renamed `SysBatteryInfoV20`)

Other Changes

- The dynamic heap has been increased in size to 96 KB.
- Storage RAM is no longer subdivided into multiple storage heaps of 64 KB each. All storage RAM on a memory card is configured as a single storage heap.
- Each flash ROM-based Palm device holds a serial number that identifies it uniquely and can be retrieved via `SysGetROMToken`. For more information, see “Retrieving the ROM Serial Number” on page 353 of the Palm OS Programmer’s Companion, vol. I.
- The Application Launcher (accessed via the silkscreen “Applications” button) is now an application, rather than a popup. The `SysAppLauncherDialog` function, which provides the API to the old popup launcher, is still present in Palm OS 3.0 for compatibility purposes, but has not been updated and generally should not be used. For more information, see “Application Launcher” on page 153 of the Palm OS Programmer’s Companion, vol. I.
- The sound manager supports MIDI sound files, adding new sounds, asynchronous playback, and other features. There are also new selectors for setting the volume preferences. For more information, see the section “System Boot and Reset” on page 337 of the Palm OS Programmer’s Companion, vol. I.
The following functions existed in the system previously, but were not documented:

- RctCopyRectangle
- RctGetIntersection
- RctInsetRectangle
- RctOffsetRectangle
- RctPtInRectangle
- RctSetRectangle

The following event type existed in the system previously, but was not previously documented:

- frmGotoEvent

### 3.1 New Feature Set

You can check that this feature set is implemented by checking that the system version is 3.1 or higher. Use this `FtrGet` call:

```c
err = FtrGet(sysFtrCreator,
             sysFtrNumROMVersion, &romVersion);
```

The `romVersion` parameter should be greater than or equal to 0x03103000, which can be constructed using the `sysMakeROMVersion` macro:

```c
sysMakeROMVersion(3,1,0,sysROMStageRelease,0)
```

### Functions

This feature set adds the following functions:

- ChrHorizEllipsis
- ChrNumericSpace
- ContrastAdjust
- FntWidthToOffset
- FtrPtrNew
- FtrPtrFree
- FtrPtrResize
- SelectOneTime
- WinDrawChar
- WinDrawTruncChars
NOTE: The PalmOSGlue.lib provides compatibility functions and macros for ChrHorizEllipsis, ChrNumericSpace, WinDrawChar, and WinDrawTruncChars. If you want to use these functions on systems that don’t have the 3.1 feature set, you can link your application with PalmOSGlue.lib. See the chapter “PalmOSGlue Library” on page 1891 for more information.

Changes to the Character Encoding

Starting in Palm OS 3.1, the character encoding used on most systems is Microsoft Windows code page 1252. Versions prior to 3.1 used an encoding that was very similar to code page 1252 but did not follow it exactly. The following changes to the character set are introduced in Palm OS 3.1:

- Some of the special Palm OS glyphs in the high ASCII range (such as the shortcut stroke and the command stroke) have been moved down into the control code range, and other characters (such as the numeric space and horizontal ellipsis) have been copied into the control range so that they’re guaranteed to exist in every encoding. For the numeric space and horizontal ellipsis, you can use the macros ChrNumericSpace and ChrHorizEllipsis to return the appropriate character regardless of the character map. In PalmOSGlue.lib, these two macros are named TxtGlueGetNumericSpaceChar and TxtGlueGetHorizEllipsisChar, respectively.

- The four playing-card characters have been moved from the high ASCII range in the standard four fonts to the 9-point Symbol font.

Other Changes in 3.1

- Palm OS 3.1 supports a new processor: the EZ Dragonball processor. This processor is compatible with the existing Dragonball processor, so your application should run
without changes as long as it doesn’t access registers or system globals directly.

If your application needs to know if it is running on an EZ Dragonball, it can check using the following code:

```c
DWord id, chip;
Word revision;
Err err;
err = FtrGet(sysFtrCreator,
    sysFtrNumProcessorID, &id);
if (!err) {
    chip = id & sysFtrNumProcessorMask;
    revision = id & 0x0ffff;
    if (chip==sysFtrNumProcessor328)
        // traditional Dragonball
    else if (chip==sysFtrNumProcessorEZ)
        // Dragonball EZ
}
```

- The constant `preferenceDataVersion` was removed and replaced with `preferenceDataVerLatest`.
- Character variables are now two bytes long. The type `WChar` defines a character variable.
- The keyDownEvent structure’s `chr` field (which contains the input character) has been changed from a `Word` to a `WChar`.
- The string manager functions `StrChr` and `StrStr` now treat buffers as characters, not arbitrary byte arrays. If you previously used these functions to search data buffers, your code may no longer work.
- The string manager function `StrToLower` can now handle any type of characters, including accented characters.
- The underline attribute of `FieldAttrType` now has support for the value 2. Previously, the only underline modes available were no underline (0) and gray underline (1). In Palm OS 3.1 and higher, the value 2 is interpreted as solid underline. The `UnderlineModeType` enum defined in `Window.h` defines the possible values for the underline attribute.
• The use of the `DmGetNextDatabaseByTypeCreator` `onlyLatestVers` parameter changed in 3.1. If `onlyLatestVers` is `true`, you only receive one matching database for each type/creator pair. In version 3.0 and earlier, you could receive multiple matching databases if `onlyLatestVers` was `true`. See that function’s description for a more detailed description.

### 3.2 New Feature Set

You can check that this feature set is implemented by checking that the system version is 3.2 or higher. Use this `FtrGet` call:

```c
err = FtrGet(sysFtrCreator,
              sysFtrNumROMVersion, &romVersion);
```

The `romVersion` parameter should be greater than or equal to `0x03203000`, which can be constructed using the `sysMakeROMVersion` macro:

```c
sysMakeROMVersion(3,2,0,sysROMStageRelease,0)
```

#### Functions

This feature set adds the following functions:

- `AlmGetProcAlarm`
- `NetLibConfigIndexFromName`
- `AlmSetProcAlarm`
- `NetLibConfigList`
- `ClipboardAppendItem`
- `NetLibConfigMakeActive`
- `DmGetDatabaseLockState`
- `NetLibConfigRename`
- `ErrAlert`
- `NetLibConfigSaveAs`
- `NetLibConfigAliasGet`
- `NetLibConfigSaveAs`
- `NetLibConfigAliasSet`
- `NetLibOpenConfig`
- `NetLibConfigDelete`
- `SndPlaySmfResource`

Note that the `NetLib...` functions, although present in the 3.2 New Feature Set, are first declared in the Palm OS 5 SDK.
Existing Functions that Changed

Two functions that existed in 3.0 were changed in 3.2:

SysGremlins was removed and replaced with a SysGremlins macro that maps it to the function HostGremlinIsRunning. The prototype is slightly different, but you can still call SysGremlins in the same way you did before.

PrgStartDialog (old function renamed PrgStartDialogV31)

Other Changes in 3.2

• The prototype for the system use only function AlmDisplayAlarm changed from no return value to a Boolean return value. This change may affect system patches and extensions that intercept AlmDisplayAlarm calls.

International Feature Set

You can check that this feature set is implemented by checking for the existence of the international manager. You can check by calling FtrGet as follows:

err = FtrGet(sysFtrCreator, sysFtrNumIntlMgr, &value);

If the international manager is installed, the value parameter will be non-zero and the returned error should also be zero (for no error).

You can learn more about the international manager by reading the chapter “Localized Applications” on page 363 in the Palm OS Programmer’s Companion, vol. I.

NOTE: If you want to use international functions on systems that don’t have the international feature, you can link your application with PalmOSGlue.lib. The functions in this library are the same as those listed below except that they use the prefix “TxtGlue” instead of “Txt.” For more information, see the chapter “PalmOSGlue Library” on page 1891.
Functions
This feature set adds the following functions:

Text Manager Functions

- `TxtByteAttr`
- `TxtCaselessCompare`
- `TxtCharAttr`
- `TxtCharBounds`
- `TxtCharEncoding`
- `TxtCharIsAlNum`
- `TxtCharIsAlpha`
- `TxtCharIsCtrl`
- `TxtCharIsDigit`
- `TxtCharIsGraph`
- `TxtCharIsHardKey`
- `TxtCharIsHex`
- `TxtCharIsLower`
- `TxtCharIsPrint`
- `TxtCharIsPunct`
- `TxtCharIsSpace`
- `TxtCharIsUpper`
- `TxtCharSize`

Other Functions

- `IntlGetRoutineAddress`

Removed Functions and Macros

If the international feature set exists, then the following functions and macros are no longer available:
Japanese Feature Set

You can check that the Japanese feature set is implemented by checking if the unit is Japanese. You can check by calling `FtrGet` as follows:

```c
err = FtrGet(sysFtrCreator, sysFtrNumEncoding, &value);
```

The unit has the Japanese OS if the `value` parameter is `charEncodingCP932`.

For further information about the Japanese implementation, see the section “Notes on the Japanese Implementation” in the Palm OS Programmer's Companion, vol. I.

Wireless Internet Feature Set

You can check that this feature set is implemented by checking for the existence of the Web Clipping Application Viewer (`Viewer`) and iMessenger™ applications. Here’s an example of how to check for Viewer:
DmSearchStateType searchState;
UInt16 cardNo;
LocalID dbID;
err = DmGetNextDatabaseByTypeCreator(true,
 &searchState, sysFileTApplication,
 sysFileCClipper, true, &cardNo, &dbID);

If Viewer is not present, the
DmGetNextDatabaseByTypeCreator routine returns an error.
To check for iMessenger, you can use the creator type
sysFileCMessaging.

**NOTE:** The Viewer was formerly described as the Clipper.

You can learn more about the Palm.Net™ system for wireless
Internet access and the programmatic interfaces to the Viewer and
iMessenger applications by reading the chapter “Internet and
Messaging Applications” in the *Palm OS Programmer’s Companion,
vol. II, Communications*. For a more complete description, see the
*Web Clipping Developer’s Guide*.

**Launch Codes**

This feature set adds the following launch codes:

- **sysAppLaunchCmdAddRecord** (for iMessenger
  application; existed for Mail in 3.0)
- **sysAppLaunchCmdGoToURL**
- **sysAppLaunchCmdOpenDB**
- **sysAppLaunchCmdURLParams**

**Events**

This feature set adds the following events:

- **inetSockReadyEvent**
- **inetSockStatusChangeEvent**

This feature set also adds the following **KeyDownEvent** key codes:
These key codes are described in the section Wireless keyDownEvent Key Codes.

### Functions

This feature set adds the following functions.

#### Internet Library Functions

For more information, see the chapter “Network Communication” in the *Palm OS Programmer’s Companion*, vol. II, Communications.

<table>
<thead>
<tr>
<th>Internet Library Functions</th>
<th>Internet Library Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>vchrHardAntenna</td>
<td>INetLibSockClose</td>
</tr>
<tr>
<td>vchrRadioCoverageOK</td>
<td>INetLibSockConnect</td>
</tr>
<tr>
<td>vchrRadioCoverageFail</td>
<td>INetLibSockHTTPAttrGet</td>
</tr>
<tr>
<td></td>
<td>INetLibSockHTTPAttrSet</td>
</tr>
<tr>
<td></td>
<td>INetLibSockHTTPReqCreate</td>
</tr>
<tr>
<td></td>
<td>INetLibSockHTTPReqSend</td>
</tr>
<tr>
<td></td>
<td>INetLibSockOpen</td>
</tr>
<tr>
<td></td>
<td>INetLibSockRead</td>
</tr>
<tr>
<td></td>
<td>INetLibSockSettingGet</td>
</tr>
<tr>
<td></td>
<td>INetLibSockSettingSet</td>
</tr>
<tr>
<td></td>
<td>INetLibSockStatus</td>
</tr>
<tr>
<td></td>
<td>INetLibURLCrack</td>
</tr>
<tr>
<td></td>
<td>INetLibURLGetInfo</td>
</tr>
<tr>
<td></td>
<td>INetLibURLOpen</td>
</tr>
<tr>
<td></td>
<td>INetLibURLsAdd</td>
</tr>
<tr>
<td></td>
<td>INetLibWiCmd</td>
</tr>
</tbody>
</table>

### New Serial Manager Feature Set

The New Serial Manager feature set has two different versions.
New Serial Manager Feature Set Version 1

You can check that this feature set is implemented by checking for the existence of the new Serial Manager. You can check by calling `FtrGet` as follows:

```c
err = FtrGet(sysFileCSerialMgr,
              sysFtrNewSerialPresent, &value);
```

If the new Serial Manager is installed, the `value` parameter will be non-zero and the returned error should also be zero (for no error).

You can learn more about the new Serial Manager and Connection Manager by reading the sections “The Serial Manager” on page 92 and “The Connection Manager” on page 116 in the *Palm OS Programmer’s Companion, vol. II, Communications*.

This feature set adds the following functions.

### Serial Manager Functions

- `SrmClearErr`
- `SrmClose`
- `SrmControl`
- `SrmGetDeviceCount`
- `SrmGetDeviceInfo`
- `SrmGetStatus`
- `SrmOpen`
- `SrmOpenBackground`
- `SrmPrimeWakeupHandler`
- `SrmReceive`
- `SrmReceiveCheck`
- `SrmReceiveFlush`
- `SrmReceiveWait`
- `SrmReceiveWindowClose`
- `SrmReceiveWindowOpen`
- `SrmSend`
- `SrmSendCheck`
- `SrmSendFlush`
- `SrmSendWait`
- `SrmSetReceiveBuffer`
- `SrmSetWakeupHandler`
- `WakeupHandlerProcPtr`
- `DrvEntryPointProcPtr`
- `GetSizeProcPtr`
- `GetSpaceProcPtr`
- `VdrvControlProcPtr`
- `VdrvOpenProcPtr`
- `VdrvStatusProcPtr`
- `VdrvWriteProcPtr`
- `WriteBlockProcPtr`
- `WriteByteProcPtr`
Connection Manager Functions

- CncAddProfile
- CncDeleteProfile
- CncGetProfileInfo
- CncGetProfileList

Serial Link Manager Function

- SlkSocketPortID

New Serial Manager Feature Set Version 2

You can check that version 2 of the new Serial Manager feature set is implemented by checking the Serial Manager version number and the Palm OS version number. You can check by calling `FtrGet` as follows:

```c
err = FtrGet(sysFileCSerialMgr,
              sysFtrNewSerialVersion, &value);
err = FtrGet(sysFtrCreator,
              sysFtrNumROMVersion, &romVersion);
```

The new Serial Manager is present if:

- Both calls to `FtrGet` return zero (for no error).
- The `value` parameter is 2.
- The `romVersion` parameter is `0x04003000`, which can be constructed using the `sysMakeROMVersion` macro:
  ```c
  sysMakeROMVersion(4,0,0,sysROMStageRelease,0)
  ```

This feature set adds the following functions.

Serial Manager Functions

- SrmCustomControl
- SrmExtOpen
- SrmExtOpenBackground

Virtual Driver Functions

- VdrvControlCustomProcPtr
- VdrvOpenProcV4Ptr
- SignalCheckPtr
IMPORTANT: Some Handspring devices ship with Palm OS version 3.5 but have new Serial Manager feature set version 2. These devices support the SignalCheckPtr virtual driver function and have expanded functionality for USB support, but they do not support the other function calls in this feature set.

You can learn more about the new Serial Manager by reading the chapter “Serial Communication” on page 89 of Palm OS Programmer’s Companion, vol. II, Communications.

Connection Manager Feature Set

You can check that the Connection Manager feature set is implemented by checking the value of the Connection Manager feature. You can check by calling FtrGet as follows:

```c
err = FtrGet(kCncFtrCncMgrCreator,
kCncFtrCncMgrVersion, &version);
```

The version parameter should be greater than or equal to 0x00040001. In the 4.0 Palm OS SDK, this value is represented by the kCncMgrVersion constant.

NOTE: An earlier version of the Connection Manager is available if New Serial Manager Feature Set Version 1 is present.

Functions

This feature set adds the following functions:

- CncProfileCloseDB
- CncProfileCount
- CncProfileCreate
- CncProfileDelete
- CncProfileGetCurrent
- CncProfileGetIDFromIndex
- CncProfileGetIDFromName
- CncProfileGetIndex
- CncProfileOpenDB
- CncProfileSetCurrent
- CncProfileSettingGet
- CncProfileSettingGetSet
3.5 New Feature Set

You can check that this feature set is implemented by checking that the system version is 3.5 or higher. Use this `FtrGet` call:

```c
err = FtrGet(sysFtrCreator,
              sysFtrNumROMVersion, &romVersion);
```

The `romVersion` parameter should be greater than or equal to 0x03503000, which can be constructed using the `sysMakeROMVersion` macro:

```c
sysMakeROMVersion(3,5,0,sysROMStageRelease,0)
```

Launch Codes

This feature set adds the following launch codes:

```c
sysAppLaunchCmdNotify
```

Events

This feature set adds the following events:

```c
frmGadgetEnterEvent
frmGadgetMiscEvent
menuCmdBarOpenEvent
menuOpenEvent
```

Functions

This feature set adds the following functions.

Bitmaps

```c
BmpBitsSize
BmpColortableSize
BmpCompress
BmpCreate
BmpDelete
```

```c
BmpGetBits
BmpGetColortable
BmpSize
ColorTableEntries
```

For more information on creating and using bitmaps, see the section “Bitmaps” on page 123 of the *Palm OS Programmer’s Companion*, vol. I.
Controls

CtlGetSliderValues  CtlSetGraphics  
CtlNewGraphicControl  CtlSetSliderValues  
CtlNewSliderControl

These functions add support for graphical buttons and slider controls. For more information, see the section “Offscreen Windows” on page 91 of the Palm OS Programmer’s Companion, vol. I.

Forms

FrmCustomResponseAlert  FrmSetGadgetHandler  
FrmNewGsi

Among the changes to form functions is extended gadget support. For more information on gadgets and extended gadgets, see the section “Custom UI Objects (Gadgets)” on page 140 of the Palm OS Programmer’s Companion, vol. I.

Menus

MenuAddItem  MenuCmdBarAddButton  
MenuCmdBarDisplay  MenuCmdBarGetButtonData  
MenuHideItem  MenuShowItem

For more information on using menu functions, see the section “Menus” on page 105 of the Palm OS Programmer’s Companion, vol. I.

Overlay Manager

OmGetCurrentLocale  OmLocaleToOverlayDBName  
OmGetIndexedLocale  OmOverlayDBNameToLocale  
OmGetRoutineAddress  OmSetSystemLocale  
OmGetSystemLocale

For more information on using the overlay manager, see the section “Using Overlays to Localize Resources” on page 365 of the Palm OS Programmer’s Companion, vol. I.

Private Records

SecSelectViewStatus  SecVerifyPW
Tables

TblGetItemPtr  TblSetColumnMasked
TblRowMasked   TblSetRowMasked

UI Colors

UIColorGetTableEntryIndex  UIColorGetTableEntry
UIColorGetTableEntryRGB

For more information on using the UI Colors API, see the section “Color and Grayscale Support” on page 144 of the Palm OS Programmer’s Companion, vol. I.

UI Controls

UIBrightnessAdjust  UIPickColor

Windows

WinCreateBitmapWindow  WinPaintRectangle
WinDrawPixel  WinPaintRectangleFrame
WinErasePixel  WinPalette
WinGetBitmap  WinPopDrawState
WinGetPatternType  WinPushDrawState
WinGetPixel  WinRGBToIndex
WinIndexToRGB  WinScreenLock
WinInvertPixel  WinScreenMode
WinPaintBitmap  WinScreenUnlock
WinPaintChar  WinSetBackColor
WinPaintChars  WinSetDrawMode
WinPaintLine  WinSetForeColor
WinPaintLines  WinSetPatternType
WinPaintPixel  WinSetTextColor
WinPaintPixels

For more information on using the window functions, see the section “Drawing on the Palm Powered Handheld” on page 72 of the Palm OS Programmer’s Companion, vol. I.
Miscellaneous New Functions

- DmOpenDBNoOverlay
- ExgDoDialog
- DateToAscii
- ResLoadConstant
- TxtParamString

Existing Functions that Changed

The following functions that existed prior to 3.5 have changed in release 3.5:

- ScrDisplayMode was changed to WinScreenMode.
- ContrastAdjust was changed to UIContrastAdjust.
- SelectTime (old function renamed SelectTimeV33)

New Data Types

The data types Byte, Word, DWord and so on are now deprecated. It is recommend that you use the corresponding new data types. For example, use Int16 instead of SWord and UInt32 instead of DWord. In particular, the unfortunate distinction between Handle/VoidHand has been fixed; use MemHandle instead.

To learn in general how the type names changed, see the header file PalmOSCompatibility.h. This file provides a mapping from the old type name to the new type name. If you need to move forward without modifying your code, you can include this file in your project to provide declarations for the old type names.

Changes to Events

- The tapCount field has been added to the EventType structure. The tapCount field specifies the number of times the user tapped the pen at the current location; in fields, two taps selects a word, and three taps selects a line.
IMPORTANT: Because the tapCount field has been added to the EventType structure, it has become more critical that you clear the event structure before using it to add a new event to the queue. Otherwise, the tapCount will be incorrect for the new event.

- The structures for ctlRepeatEvent and ctlSelectEvent have a value field added to them. This new field is used only for sliders; it holds the current value of the slider.
- Form objects now handle the frmTitleSelectEvent by adding a keyDownEvent with the vchrMenu character to the event queue (which causes the form’s menu to display).
- Some of the structure definitions for system-level events have moved from Event.h to SysEvent.h.
- The winEnterEvent is now not generated until FrmDrawForm is called. Make sure to draw your form in response to frmOpenEvent, not winEnterEvent.
- EvtSetNullEventTick is now a function. In previous releases it was a macro.

Other Changes

- FrmDrawForm
  On release 3.5, FrmDrawForm erases the window’s rectangle before it draws, so you must perform custom drawing after the call to FrmDrawForm, not before. If you have drawn before the call to FrmDrawForm, your changes are lost. On debug ROMs, the window handle is invalid until FrmDrawForm is called so that draws before FrmDrawForm result in a bus error.

- Resource Manager
  The resource manager functions have been updated to work with overlay databases. See “Using Overlays to Localize Resources” on page 365 of the Palm OS Programmer’s Companion, vol. I.
• **DmGetDatabase**
  The order in which this call returns databases has changed. Previously all of the databases from ROM were returned first, then all from RAM. Now they are intermingled. Developers should not rely on the order in which databases are returned from this call.

• **StrToLower**
  This function is different in 3.5 Latin ROMs. Previously it only changed A through Z. Now it also changes high ASCII characters.

• **Time Manager**
  If you are using a debug ROM, the string buffer is filled with `dateStringLength` or `longStrLength` debugging bytes, depending on the `dateFormat` parameter. For the routines that return the day-of-week name in addition to the date, the size of the buffers has been expanded, so developers need to check the max lengths defined in `DateTime.h`.

• The format of the storage heap header has changed, thus any existing saved Simulator card images are invalid and should be tossed.

• **Category Data Structures**
  The data structure `AppInfoType` has been documented. `CategoryCreateList` now has a “hide” function with two new constants; `categoryHideEditCategory`, and `categoryDefaultEditCategoryString`.

• **FtrPtrNew**
  `FtrPtrNew` now allows allocating chunks larger than 64KB.

• **Dynamic heap**
  The dynamic heap is now sized based on the amount of memory available to the system.
### Notification Feature Set

You can check that this feature set is implemented by checking for the existence of the notification manager. You can check by calling `FtrGet` as follows:

```c
err = FtrGet(sysFtrCreator, sysFtrNumNotifyMgrVersion, &value);
```

If the notification manager is part of the system, the `value` parameter will be non-zero and the returned error should also be zero (for no error).

#### Notification Manager

- `SysNotifyBroadcast`
- `SysNotifyBroadcastDeferred`
- `SysNotifyRegister`
- `SysNotifyUnregister`

To learn more about the notification manager, see the section “Notifications” on page 30 of the *Palm OS Programmer’s Companion*, vol. I.

### 4.0 New Feature Set

You can check that this feature set is implemented by checking that the system version is 4.0 or higher. Use this `FtrGet` call:

```c
err = FtrGet(sysFtrCreator, sysFtrNumROMVersion, &romVersion);
```

The `romVersion` parameter should be greater than or equal to `0x04003000`, which can be constructed using the `sysMakeROMVersion` macro.

---

**Table B.1 Dynamic heap size**

<table>
<thead>
<tr>
<th>Device RAM Size</th>
<th>Heap Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x &lt; 2$MB</td>
<td>64KB</td>
</tr>
<tr>
<td>$2$MB $\leq x &lt; 4$MB</td>
<td>128KB</td>
</tr>
<tr>
<td>$x \geq 4$MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>
sysMakeROMVersion(4, 0, 0, sysROMStageRelease, 0)

Launch Codes
This feature set adds the following launch codes:

sysAppLaunchCmdAttention
sysAppLaunchCmdExgGetData
sysAppLaunchCmdExgPreview

Notifications
This feature set adds the following notifications:

cncNotifyProfileEvent
sysExternalConnectorAttachEvent
sysExternalConnectorDetachEvent
sysNotifyCardInsertedEvent
sysNotifyCardRemovedEvent
sysNotifyDBDeletedEvent
sysNotifyDeleteProtectedEvent
sysNotifyDeviceUnlocked
sysNotifyGotUsersAttention
sysNotifyHelperEvent
sysNotifyLocaleChangedEvent
sysNotifyNetLibIFMediaEvent
sysNotifyRetryEnqueueKey
sysNotifyVolumeMountedEvent
sysNotifyVolumeUnmountedEvent

Functions
This feature set adds the following functions:

Attention Manager

AttnDoSpecialEffects  AttnIndicatorEnable
AttnForgetIt          AttnIndicatorEnabled
AttnGetAttention      AttnIterate
AttnGetCounts         AttnListOpen
                      AttnUpdate
Date and Time Manager

TimeZoneToAscii
TimTimeZoneToUTC
TimUTCToTimeZone

Exchange Manager

ExgGetTargetApplication  ExgControl
ExgGetDefaultApplication  ExgNotifyGoto
ExgGetRegisteredApplications  ExgNotifyPreview
ExgSetDefaultApplication  ExgRequest

For more information on the Exchange Manager, see the chapter Chapter 1, “Object Exchange,” on page 1 of the Palm OS Programmer’s Companion, vol. II, Communications.

Exchange Library

ExqLibAccept  ExqLibOpen
ExqLibClose  ExqLibPut
ExqLibConnect  ExqLibReceive
ExqLibControl  ExqLibRequest
ExqLibDisconnect  ExqLibSend
ExqLibGet  ExqLibSleep
ExqLibHandleEvent  ExqLibWake


Locale Manager

LmGetLocaleSetting  LmGetNumLocales
LmLocaleToIndex

Miscellaneous UI

PhoneNumberLookupCustom
Notification Manager

SysNotifyBroadcastFromInterrupt

PDI Library Functions

PdiDefineReaderDictionary  PdiSetEncoding
PdiDefineResizing         PdiWriteBeginObject
PdiEnterObject            PdiWriteEndObject
PdiLibClose               PdiWriteParameter
PdiLibOpen                PdiWriteParameterStr
PdiParameterPairTest      PdiWriteProperty
PdiReaderDelete           PdiWritePropertyBinaryValue
PdiReaderNew              PdiWritePropertyFields
PdiReadParameter          PdiWritePropertyStr
PdiReadProperty           PdiWritePropertyValue
PdiReadPropertyField      PdiWriterDelete
PdiReadPropertyName       PdiWriterNew

For more information, see the chapter Chapter 3, “Personal Data Interchange,” in Palm OS Programmer’s Companion, vol. II, Communications.

Sound Manager Functions

SndInterruptSmfIrregardless  SndPlaySmfIrregardless
SndPlaySmfResourceIrregardless

For more information, see the chapter Chapter 10, “Palm System Support,” in Palm OS Programmer’s Companion, vol. I.
Telephony Manager Functions

TelCancel
TelCfgGetPhoneNumber
TelCfgGetSmsCenter
TelClose
TelClosePhoneConnection
TelDtcCallNumber
TelDtcCloseLine
TelDtcReceiveData
TelDtcSendData
TelEmcCall
TelEmcCloseLine
TelEmcGetNumber
TelEmcGetNumberCount
TelEmcGetNumberCount
TelEmcSelectNumber
TelEmcSetNumber
TelGetCallState
TelGetEvent
TelGetTelephonyEvent
TelInfGetInformation
TelIs<FunctionName>Supported
TelIs<ServiceSet>Available
TelIsCfgServiceAvailable
TelIsDtcServiceAvailable
TelIsEmcServiceAvailable
TelIsInfServiceAvailable
TelIsNwkServiceAvailable
TelIsOemServiceAvailable
TelIsPhbServiceAvailable
TelIsPhoneConnected
TelIsPoseServiceAvailable
TelIsSmsServiceAvailable
TelIsSndServiceAvailable
TelIsSpcServiceAvailable
TelIsStyServiceAvailable
TelMatchPhoneDriver
TelNwkGetLocation
TelNwkGetNetworkName
TelNwkGetNetworks
TelNwkGetNetworkType
TelNwkGetSearchMode
TelNwkGetSelectedNetwork
TelNwkGetSignalLevel
TelNwkSelectNetwork
TelNwkSetSearchMode
TelOemCall
TelOpen
TelOpenPhoneConnection
TelPhbAddEntry
TelPhbDeleteEntry
TelPhbGetAvailablePhonebooks
TelPhbGetEntries
TelPhbGetEntry
TelPhbGetEntryCount
TelPhbGetEntryMaxSizes
TelPhbGetSelectedPhonebook
TelPhbSelectPhonebook
TelPowGetBatteryStatus
TelPowGetPowerLevel
TelPowSetPhonePower
TelSendCommandString
Compatibility Guide
4.0 New Feature Set

### Palm OS Programmer’s API Reference

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TelSmsDeleteMessage</code></td>
<td><code>TelSndPlayKeyTone</code></td>
</tr>
<tr>
<td><code>TelSmsGetAvailableStorage</code></td>
<td><code>TelSndStopKeyTone</code></td>
</tr>
<tr>
<td><code>TelSmsGetDataMaxSize</code></td>
<td><code>TelSpcAcceptCall</code></td>
</tr>
<tr>
<td><code>TelSmsGetMessageCount</code></td>
<td><code>TelSpcCallNumber</code></td>
</tr>
<tr>
<td><code>TelSmsGetSelectedStorage</code></td>
<td><code>TelSpcCloseLine</code></td>
</tr>
<tr>
<td><code>TelSmsGetUniqueId</code></td>
<td><code>TelSpcConference</code></td>
</tr>
<tr>
<td><code>TelSmsReadMessage</code></td>
<td><code>TelSpcGetCallerNumber</code></td>
</tr>
<tr>
<td><code>TelSmsReadMessages</code></td>
<td><code>TelSpcHoldLine</code></td>
</tr>
<tr>
<td><code>TelSmsReadReport</code></td>
<td><code>TelSpcPlayDTMF</code></td>
</tr>
<tr>
<td><code>TelSmsReadReports</code></td>
<td><code>TelSpcRejectCall</code></td>
</tr>
<tr>
<td><code>TelSmsReadSubmittedMessage</code></td>
<td><code>TelSpcRetrieveHeldLine</code></td>
</tr>
<tr>
<td><code>TelSmsReadSubmittedMessages</code></td>
<td><code>TelSpcSelectLine</code></td>
</tr>
<tr>
<td><code>TelSmsSelectStorage</code></td>
<td><code>TelSpcSendBurstDTMF</code></td>
</tr>
<tr>
<td><code>TelSmsSendManualAcknowledge</code></td>
<td><code>TelSpcStartContinuousDTMF</code></td>
</tr>
<tr>
<td><code>TelSmsSendMessage</code></td>
<td><code>TelSpcStopContinuousDTMF</code></td>
</tr>
<tr>
<td><code>TelSndMute</code></td>
<td><code>TelSpcTranparentAuthCode</code></td>
</tr>
<tr>
<td><code>TelSndPlayKeyTone</code></td>
<td><code>TelStyEnterAuthenticationCode</code></td>
</tr>
<tr>
<td><code>TelSndStopKeyTone</code></td>
<td><code>TelStyGetAuthenticationState</code></td>
</tr>
<tr>
<td><code>TelSndMute</code></td>
<td></td>
</tr>
</tbody>
</table>

For more information about the Telephony Manager, see Chapter 10, “Telephony Manager,” in Palm OS Programmer’s Companion, vol. II, Communications.

### Windows Manager

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>WinGetPixelRGB</code></td>
<td><code>WinSetBackColorRGB</code></td>
</tr>
<tr>
<td><code>WinSetForeColorRGB</code></td>
<td><code>WinSetTextColorRGB</code></td>
</tr>
</tbody>
</table>

### Internationalization Functions

The following functions have been added to aid with creating localized applications:

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FntWCharWidth</code></td>
<td><code>OmGetNextSystemLocale</code></td>
</tr>
<tr>
<td><code>StrCompareAscii</code></td>
<td><code>TxtConvertEncoding</code></td>
</tr>
<tr>
<td><code>TxtGetWordWrapOffset</code></td>
<td><code>TxtNameToEncoding</code></td>
</tr>
<tr>
<td><code>IntlSetRoutineAddress</code></td>
<td></td>
</tr>
</tbody>
</table>
Existing Functions that Changed

The following functions that existed prior to 4.0 have changed in release 4.0:

- `ExgRegisterData` has been deprecated. The preferred function is now `ExgRegisterDatatype`.

- `ExgGet` was previously unimplemented. It has been implemented in release 4.0

- `ExgNotifyReceive` was previously a private function. It is now a public function, intended to be used by exchange libraries.

- `FldRecalculateField` now updates the word-wrapping information whenever `FldRecalculateField` is called, regardless of the value of the `redraw` parameter. Prior to Palm OS 4.0 it updated the word-wrapping information only if the `redraw` parameter was set to `true`.

- `StrNCaselessCompare` now requires both of its string parameters to be null-terminated.

- `StrNCompare` now requires both of its string parameters to be null-terminated.

Expansion Manager Feature Set

Because not every system has (or needs) Expansion Manager services, applications wishing to use these services should check to make sure they are present before calling them. This is accomplished by checking for the Expansion Manager’s system feature with a call to `FtrGet`, supplying `sysFileCExpansionMgr` for the feature creator and `expFtrIDVersion` for the feature number.

The following code shows how to check for the presence and proper version of the Expansion Manager. Note that `expectedExpMgrVersionNum` should be replaced by the actual version number you expect.
compatibility guide
vfs manager feature set

uint32 expMgrVersion;
err err;

er = FtrGet(sysFileCExpansionMgr,
    expFtrIDVersion, &expMgrVersion);
if(err){
    // Expansion Manager not installed
} else {
    // check version number of Expansion Manager,
    // if necessary
    if(expMgrVersion == expectedExpMgrVersionNum)
        // everything is OK
}

you can learn more about the expansion manager by reading

functions
this feature set adds the following functions.

expansion manager functions
expCardGetSerialPort    expSlotDriverInstall
expCardInfo            expSlotDriverRemove
expCardPresent         expSlotEnumerate
expSlotDriverInstall   expSlotLibFind

vfs manager feature set
because not every system has (or needs) virtual file system (vfs)
manager services, applications wishing to use these services should
check to make sure they are present before calling them. this is
accomplished by checking for the vfs manager’s system feature
with a call to ftrGet, supplying sysFileCVFSMgr for the feature
creator and vfsFtrIDVersion for the feature number.

the following code shows how to check for the presence and proper
version of the vfs manger. note that
expectedVFSMgrVersionNum should be replaced by the actual
version number you expect.
$$\text{UInt32 vfsMgrVersion;}$$
$$\text{Err err;}$$

$$\text{err = FtrGet(sysFileCVFSMgr,}$$
$$\text{vfsFtrIDVersion, &vfsMgrVersion);}$$
$$\text{if(err){}$$
$$\text{ // VFS Manager not installed}$$
$$\text{}}$$
$$\text{else { }$$
$$\text{ // check version number of VFS Manager,}$$
$$\text{ // if necessary}$$
$$\text{if(vfsMgrVersion == expectedVFSMgrVersionNum)}$$
$$\text{ // everything is OK}$$

You can learn more about the VFS manager by reading Chapter 53, “Virtual File System Manager,” on page 1075.

**Functions**

This feature set adds the following functions.

**VFS Manager Functions**

- `VFSCustomControl`
- `VFSFileSize`
- `VFSCustomControl`
- `VFSFileTell`
- `VFSDirEntryEnumerate`
- `VFSFileWrite`
- `VFSExportDatabaseToFile`
- `VFSGetDefaultDirectory`
- `VFSExportDatabaseToFileCustom`
- `VFSImportDatabaseFromFile`
- `VFSFileClose`
- `VFSImportDatabaseFromFileCustom`
- `VFSFileCreate`
- `VFSInstallFSLib`
- `VFSFileDBGetRecord`
- `VFSRegisterDefaultDirectory`
- `VFSFileDBGetResource`
- `VFSRemoveFSLib`
Because not every system has (or needs) Bluetooth Library services, applications wishing to use these services should check to make sure they are present before calling them. This is accomplished by checking for the Bluetooth Library’s system feature with a call to FtrGet, supplying btLibFeatureCreator for the feature creator and btLibFeatureVersion for the feature number.

The following code shows how to check for the presence of the Bluetooth Library.

```c
UInt32 btVersion;

// Make sure Bluetooth components are installed
// This check also ensures Palm OS 4.0 or greater
if (FtrGet(btLibFeatureCreator, btLibFeatureVersion, &btVersion) != errNone) {
    // Alert the user if it's the active application
```

蓝牙库功能集

因为不是每个系统都（或需要）蓝牙库服务，应用程序需要使用这些服务时应该检查它们是否已经存在。这可以通过检查蓝牙库的系统特征来实现，使用FtrGet函数，提供btLibFeatureCreator作为特征创建者和btLibFeatureVersion作为特征编号。

以下代码展示了如何检查蓝牙库的 Presence.

```c
UInt32 btVersion;

// Make sure Bluetooth components are installed
// This check also ensures Palm OS 4.0 or greater
if (FtrGet(btLibFeatureCreator, btLibFeatureVersion, &btVersion) != errNone) {
    // Alert the user if it's the active application
```
if ((launchFlags & sysAppLaunchFlagNewGlobals) &&
    (launchFlags & sysAppLaunchFlagUIApp))
    FrmAlert (MissingBtComponentsAlert);
    return sysErrRomIncompatible;
}

You can learn more about the Bluetooth Library by reading Chapter 6, “Bluetooth,” on page 131 of the Palm OS Programmer’s Companion, vol. II, Communications.

Functions
This feature set adds the following functions.

**Bluetooth Library Security Functions**

- `BtLibSecurityFindTrustedDeviceRecord`
- `BtLibSecurityNumTrustedDeviceRecords`
- `BtLibSecurityGetTrustedDeviceRecordInfo`
- `BtLibSecurityRemoveTrustedDeviceRecord`

**Bluetooth Library Utility Functions and Macros**

- `BtLibAddrAToBtd`
- `BtLibAddrBtdToA`
- `BtLibL2CapHToNL`
- `BtLibL2CapHToNS`
- `BtLibL2CapNToHL`
- `BtLibL2CapNToHS`
- `BtLibRfCommHToNL`
- `BtLibRfCommHToNS`
- `BtLibRfCommNToHL`
- `BtLibRfCommNToHS`

**Bluetooth Library Management Functions**

- `BtLibClose`
- `BtLibOpen`
- `BtLibCancelInquiry`
- `BtLibLinkSetState`
- `BtLibPiconetCreate`
- `BtLibPiconetDestroy`
<table>
<thead>
<tr>
<th>Bluetooth Library Feature Set Functions</th>
<th>Bluetooth Library Service Discovery Protocol Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>BtLibDiscoverMultipleDevices</em></td>
<td><em>BtLibSdpCompareUuids</em></td>
</tr>
<tr>
<td><em>BtLibDiscoverSingleDevice</em></td>
<td><em>BtLibSdpGetPSMByUuid</em></td>
</tr>
<tr>
<td><em>BtLibGetGeneralPreference</em></td>
<td><em>BtLibSdpGetRawDataElementType</em></td>
</tr>
<tr>
<td><em>BtLibGetRemoteDeviceName</em></td>
<td><em>BtLibSdpGetRawDataElementType</em></td>
</tr>
<tr>
<td><em>BtLibGetSelectedDevices</em></td>
<td><em>BtLibSdpGetRawDataElementType</em></td>
</tr>
<tr>
<td><em>BtLibLinkConnect</em></td>
<td><em>BtLibSdpMapRemote</em></td>
</tr>
<tr>
<td><em>BtLibLinkDisconnect</em></td>
<td><em>BtLibSdpGetSizeOfRaw Attribute</em></td>
</tr>
<tr>
<td><em>BtLibLinkGetState</em></td>
<td><em>BtLibSdpGetSizeOfStringOrUrlLength</em></td>
</tr>
<tr>
<td><em>BtLibManagementCallback</em></td>
<td><em>BtLibSdpGetSizeOfStringOrUrlLength</em></td>
</tr>
<tr>
<td><em>BtLibRegisterManagementNotification</em></td>
<td><em>BtLibSdpGetSizeOfStringOrUrlLength</em></td>
</tr>
<tr>
<td><em>BtLibPrint</em></td>
<td><em>BtLibSdpGetSizeOfStringOrUrlLength</em></td>
</tr>
<tr>
<td><em>BtLibRead</em></td>
<td><em>BtLibSdpGetSizeOfStringOrUrlLength</em></td>
</tr>
<tr>
<td><em>BtLibSetGeneralPreference</em></td>
<td><em>BtLibSdpGetSizeOfStringOrUrlLength</em></td>
</tr>
<tr>
<td><em>BtLibStartInquiry</em></td>
<td><em>BtLibSdpGetSizeOfStringOrUrlLength</em></td>
</tr>
<tr>
<td><em>BtLibStopInquiry</em></td>
<td><em>BtLibSdpGetSizeOfStringOrUrlLength</em></td>
</tr>
<tr>
<td><em>BtLibUnregisterManagementNotification</em></td>
<td><em>BtLibSdpGetSizeOfStringOrUrlLength</em></td>
</tr>
<tr>
<td><em>BtLibUnregisterManagementNotification</em></td>
<td><em>BtLibSdpGetSizeOfStringOrUrlLength</em></td>
</tr>
</tbody>
</table>

**Bluetooth Library Socket Functions**

- *BtLibSocketAdvanceCredit*
- *BtLibSocketClose*
- *BtLibSocketConnect*
- *BtLibSocketCreate*
- *BtLibSocketGetInfo*
- *BtLibSocketGetListen*
- *BtLibSocketGetSelectedDevices*
- *BtLibSocketGetSocketConnection*
- *BtLibSocketListen*
- *BtLibSocketRespondToConnection*
- *BtLibSocketSend*
### Compatibility Guide

#### High-Density Display Feature Set

You can verify that this feature set is implemented by checking the version of the Window Manager. If the Window Manager version is 4 or greater, the High-Density Display feature set is supported. To check the version of the Window Manager, use this call:

```c
err = FtrGet(sysFtrCreator,
              sysFtrNumWinVersion, &version);
```

Upon return, if `version` has a value of 4 or greater, the High-Density Display feature set is present. Note that just because the High-Density Display feature set is present, it isn’t necessarily being used. You may want to check the density of the screen, as follows:

```c
WinScreenGetAttribute(winScreenDensity, &attr);
if (attr == kDensityDouble) {
    // the screen is high density
}
```

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BtLibSdpGetServerChannelByUuid</td>
<td></td>
</tr>
<tr>
<td>BtLibSdpParseRawDataElement</td>
<td></td>
</tr>
<tr>
<td>BtLibSdpServiceRecordCreate</td>
<td></td>
</tr>
<tr>
<td>BtLibSdpServiceRecordDestroy</td>
<td></td>
</tr>
<tr>
<td>BtLibSdpServiceRecordGetAttribute</td>
<td></td>
</tr>
<tr>
<td>BtLibSdpServiceRecordGetNumListEntries</td>
<td></td>
</tr>
<tr>
<td>BtLibSdpServiceRecordGetNumLists</td>
<td></td>
</tr>
<tr>
<td>BtLibSdpServiceRecordGetRawAttribute</td>
<td></td>
</tr>
<tr>
<td>BtLibSdpServiceRecordSetAttributesForSocket</td>
<td></td>
</tr>
<tr>
<td>BtLibSdpServiceRecordSetRawAttribute</td>
<td></td>
</tr>
<tr>
<td>BtLibSdpServiceRecordsGetByServiceClass</td>
<td></td>
</tr>
<tr>
<td>BtLibSdpServiceRecordStartAdvertising</td>
<td></td>
</tr>
<tr>
<td>BtLibSdpServiceRecordStopAdvertising</td>
<td></td>
</tr>
<tr>
<td>BtLibSdpUuidInitialize</td>
<td></td>
</tr>
<tr>
<td>BtLibSdpVerifyRawDataElement</td>
<td></td>
</tr>
<tr>
<td>BtLibSocketCallback</td>
<td></td>
</tr>
</tbody>
</table>
The **5.0 New Feature Set** incorporates all of the functionality present in the High-Density Display Feature Set, so if your application is running on Palm OS 5 you can assume that the High-Density Display Feature Set is present as well.

## New Data Types
This feature set adds the following new data types.

### Bitmap Data Types
- **BitmapType**
- **BitmapTypeV0**
- **BitmapTypeV1**
- **BitmapTypeV2**
- **BitmapTypeV3**
- **DensityType**
- **PixelFormatType**

In addition, the definition of **BitmapType** changed.

### Font Data Types
- **FontDensityType**
- **FontTypeV2**

In addition, this feature set defines a new **Extended Font Resource**.

### Window Constants
This feature set adds a new set of **Window Coordinate System Constants**.

## Functions
This feature set adds the following functions.

### Bitmap Functions
- **BmpCreateBitmapV3**
- **BmpGetCompressionType**
- **BmpGetDensity**
- **BmpGetNextBitmapAnyDensity**
- **BmpGetTransparentColor**
- **BmpGetVersion**
- **BmpSetDensity**
- **BmpSetTransparentColor**
Sound Stream Feature Set

The Sound Stream feature set adds a number of “stream” functions and constants to the Sound Manager. You can verify that this feature set is supported by checking for the Sound Manager’s version number. If the Sound Manager’s version feature is defined, the Sound Stream feature set is supported.

The following code shows how to check for the presence and proper version of the Sound Manger. Note that expectedSndMgrVersionNum should be replaced by the actual version number you expect (typically, sndMgrVersionNum).

```c
UInt32 version;
Err err;

err = FtrGet(sysFileCSoundMgr, sndFtrIDVersion, &version);
if(err){
    // Sound Stream Feature Set not present
} else {
    // The Sound Stream Feature Set is present.
    // Check version number of Sound Manager,
    // if necessary
    if(version == expectedSndMgrVersionNum)
        // everything is OK
}
```
Sound Stream Data Structures and Types
This feature set adds the following data structures:

- `SndPtr`
- `SndSampleType`
- `SndStreamMode`
- `SndStreamRef`
- `SndStreamWidth`

Sound Stream Enums and Constants
This feature set adds the following enums and constants:

**Sound Stream Enums and Constants**

- `SndSampleTypeTag`
- `SndStreamModeTag`
- `SndStreamWidthTag`
- `Stereo_Pan_Constants`
- `Volume_Constants`
- `Sound_Resource_Playback_FLAGS`

Sound Stream Functions
This feature set adds the following functions:

**Sound Stream Functions**

- `SndPlayResource`
- `SndStreamCreate`
- `SndStreamDelete`
- `SndStreamGetPan`
- `SndStreamGetVolume`
- `SndStreamPause`
- `SndStreamSetPan`
- `SndStreamSetVolume`
- `SndStreamStart`
- `SndStreamStop`

In addition, the Sound Stream Feature Set defines the following callback function:

- `SndStreamBufferCallback`
5.0 New Feature Set

You can check that this feature set is implemented by checking that the system version is 5.0 or higher. Use this `FtrGet` call:

```c
err = FtrGet(sysFtrCreator, sysFtrNumROMVersion, &romVersion);
```

The `romVersion` parameter should be greater than or equal to `0x05003000`, which can be constructed using the `sysMakeROMVersion` macro:

```c
sysMakeROMVersion(5, 0, 0, sysROMStageRelease, 0)
```

This feature set corresponds to version 5.0 of Palm OS 5.

The Palm OS Application Compatibility Environment (PACE) has its own version associated with it. You can obtain this version number with:

```c
err = FtrGet('pace', 0, &paceVersion);
```

Notifications

This feature set adds the following database-related notifications:

- `sysNotifyDBCreatedEvent`
- `sysNotifyDBDirtyEvent`
- `sysNotifyDBChangedEvent`

It also broadcasts the following notifications:

- `sysNotifyAppLaunchingEvent`
- `sysNotifyAppQuittingEvent`
- `sysNotifyEventDequeuedEvent`
- `sysNotifyIdleTimeEvent`
- `sysNotifyInsPtEnableEvent`
- `sysNotifyKeyboardDialogEvent`
- `sysNotifyProcessPenStrokeEvent`
- `sysNotifyVirtualCharHandlingEvent`

Functions

This feature set adds the following functions.
ARM-Native Functions

PceNativeCall

Functions and Traps not Supported by PACE

For various reasons a number of functions and traps are not supported by PACE. The following sections list those functions, grouped according to the reason that they are not implemented.

Unimplemented “System Use Only” Functions

The following functions, which are documented as “System Use Only,” are not supported by the Palm OS Application Compatibility Environment (PACE).
As a rule, functions and traps that are not documented should be treated as if they are unimplemented and should not be used by applications.

1. Implemented on release ROMs but flagged as illegal on debug ROMs.
Implemented “System Use Only” Functions and Traps

Although marked “System Use Only,” a number of functions and traps are required by applications in the ROM, by Palm Debugger, by test applications, by scripting, or by some popular applications. Because of this, the following System Use Only functions and traps are supported by PACE. Because they are intended only for system use, however, applications should do what they can to avoid using them.

AlmEnableNotification HwrDockStatus
AttnAllowClose HwrLEDAttributes
AttnEnableNotification HwrMemReadable
AttnIndicatorAllow HwrMemWritable
AttnIndicatorAllowed HwrVibrateAttributes
AttnIndicatorGetBlinkPattern MemChunkNew
MemHeapPtr
AttnIndicatorSetBlinkPattern PenRawToScreen
PenScreenToRaw
AttnReopen SysGetAppInfo
DmResetRecordStates SysLaunchConsole
EvtEnqueuePenPoint SndSetDefaultVolume
EvtGetSilkscreenAreaList SysLCDContrast
FileReadLow SysSetA5
Find SysSetPerformance
FrmActiveState SysSleep
FrmHandleEvent SysUIBusy
HwrDelay WinDrawWindowFrame

Obsolete Functions and Traps

The following functions are not supported by PACE because they are obsolete.

FplAdd¹ FplFToA¹
FplAToF¹ FplLongToFloat¹
FplBase10Info¹ FplMul¹
FplDiv¹ FplSub¹
FplFloatToLong¹ FplFloatToULong¹
WiCmdV32

¹. Implemented on release ROMs but flagged as illegal on debug ROMs.
‘NOP’ Functions and Traps

These functions and traps should not be called by applications (many are documented as “System Use Only”). Because some third-party applications do call them, for backward compatibility they act as NOPs.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FplFree</td>
<td></td>
</tr>
<tr>
<td>FplInit</td>
<td></td>
</tr>
<tr>
<td>HwrEnableDataWrites</td>
<td></td>
</tr>
<tr>
<td>HwrDisableDataWrites</td>
<td></td>
</tr>
<tr>
<td>HwrTimerSleep</td>
<td></td>
</tr>
<tr>
<td>HwrTimerWake</td>
<td></td>
</tr>
<tr>
<td>KeyResetDoubleTap</td>
<td></td>
</tr>
<tr>
<td>KeySleep</td>
<td></td>
</tr>
<tr>
<td>KeyWake</td>
<td></td>
</tr>
<tr>
<td>PenSleep</td>
<td></td>
</tr>
<tr>
<td>PenWake</td>
<td></td>
</tr>
<tr>
<td>SerReceiveISP</td>
<td></td>
</tr>
<tr>
<td>SrmSleep</td>
<td></td>
</tr>
<tr>
<td>SrmWake</td>
<td></td>
</tr>
<tr>
<td>SysDisableInts</td>
<td></td>
</tr>
<tr>
<td>SysRestoreStatus</td>
<td></td>
</tr>
<tr>
<td>TimHandleInterrupt</td>
<td></td>
</tr>
<tr>
<td>TimSleep</td>
<td></td>
</tr>
<tr>
<td>TimWake</td>
<td></td>
</tr>
<tr>
<td>WinDisableWindow</td>
<td></td>
</tr>
<tr>
<td>WinEnableWindow</td>
<td></td>
</tr>
<tr>
<td>WinInitializeWindow</td>
<td></td>
</tr>
</tbody>
</table>

Unimplemented Rare Functions and Traps

These functions and traps are only used internally by Palm OS, by serial drivers, by OEM extensions, and the like. They are not implemented by PACE.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConGetS</td>
<td></td>
</tr>
<tr>
<td>ConPutS</td>
<td></td>
</tr>
<tr>
<td>DayDrawDays</td>
<td></td>
</tr>
<tr>
<td>DayDrawDaySelector</td>
<td></td>
</tr>
<tr>
<td>DbgCommSettings</td>
<td></td>
</tr>
<tr>
<td>DbgGetMessage</td>
<td></td>
</tr>
<tr>
<td>DlkDispatchRequest</td>
<td></td>
</tr>
<tr>
<td>DlkStartServer</td>
<td></td>
</tr>
<tr>
<td>DmMoveOpenDBContext</td>
<td></td>
</tr>
<tr>
<td>DmOpenDBWithLocale</td>
<td></td>
</tr>
<tr>
<td>FlashCompress</td>
<td></td>
</tr>
<tr>
<td>FlashErase</td>
<td></td>
</tr>
<tr>
<td>FlashProgram</td>
<td></td>
</tr>
<tr>
<td>IntlSetRoutineAddress</td>
<td></td>
</tr>
<tr>
<td>MemGetRomNVParams</td>
<td></td>
</tr>
<tr>
<td>MemNVParams</td>
<td></td>
</tr>
<tr>
<td>OEMDispatch</td>
<td></td>
</tr>
<tr>
<td>ResLoadForm</td>
<td></td>
</tr>
<tr>
<td>SlkSetSocketListener</td>
<td></td>
</tr>
<tr>
<td>SysNotifyDatabaseAdded</td>
<td></td>
</tr>
<tr>
<td>SysNotifyDatabaseRemoved</td>
<td></td>
</tr>
<tr>
<td>SysSetTrapAddress</td>
<td></td>
</tr>
</tbody>
</table>

1. Supported if the OEM supports the trap.
5.1 New Feature Set

You can check that this feature set is implemented by checking that the system version is 5.1 or higher. Use this `FtrGet` call:

```c
err = FtrGet(sysFtrCreator,
              sysFtrNumROMVersion, &romVersion);
```

The `romVersion` parameter should be greater than or equal to 0x05103000, which can be constructed using the `sysMakeROMVersion` macro:

```c
sysMakeROMVersion(5,1,0,sysROMStageRelease,0)
```

This feature set corresponds to version 5.1 of Palm OS 5.

Net Library Interface Settings

This feature set adds the following Net Library Interface Settings:

- `DriverVersion`
- `FirmwareVersion`
- `FirmwareDate`
- `80211Device`
- `80211ESSID`
- `80211AccessPointBSSID`
- `80211AssociationStatus`
- `80211MKKCallSign`
- `80211CountryTest`

See [NetLibIFSettingGet](#) for a description of each of the above settings.

CPM Library

This feature set includes the Cryptographic Provider Manager (CPM) shared library. The CPM library isn’t automatically loaded upon system boot: before making use of the CPM library you must first load it, using `SysLibFind` and `SysLibLoad`.

The CPM library defines a number of constants:

- AP Capability Constants
- Block Encryption Mode Constants
- Cipher Algorithm Constants
- Export Encoding Constants
- Hashing Algorithm Constants
• **Key Class Constants**
• **Key Usage Constants**
• **Plaintext Padding Constants**

The following structures are defined as part of the CPM library:

- APCipherInfoStruct
- APHashInfoStruct
- APKeyInfoStruct
- APProviderContextStruct
- APProviderInfoStruct
- APVerifyInfoStruct
- CPMInfoStruct

Finally, the CPM library contains the following functions:

- CPMLibDecrypt
- CPMLibDecryptFinal
- CPMLibDecryptInit
- CPMLibDecryptUpdate
- CPMLibEncrypt
- CPMLibEncryptFinal
- CPMLibEncryptInit
- CPMLibEncryptUpdate
- CPMLibExportCipherInfo
- CPMLibExportHashInfo
- CPMLibExportKeyInfo
- CPMLibExportVerifyInfo
- CPMLibGenerateKey
- CPMLibGetInfo
- CPMLibGetProviderInfo
- CPMLibHash
- CPMLibHashFinal
- CPMLibHashInit
- CPMLibImportCipherInfo
- CPMLibImportHashInfo
- CPMLibImportKeyInfo
- CPMLibImportVerifyInfo
- CPMLibReleaseCipherInfo
- CPMLibReleaseHashInfo
- CPMLibReleaseKeyInfo
- CPMLibReleaseVerifyInfo
- CPMLibVerify
- CPMLibVerifyFinal
- CPMLibVerifyInit
- CPMLibVerifyUpdate

### SSL Library

The 5.1 New Feature Set includes the Secure Sockets Layer (SSL) shared library. The SSL library isn’t automatically loaded upon system boot: before making use of the SSL library you must first load it and open it, using code similar to the following:

```c
Err error;
UInt16 libRef;

if ( SyLibFind( kSslDBName, &libRef ) != 0 )
{
    error = SysLibLoad(kSslLibType, kSslLibCreator, &libRef);
}
```
/* error checking goes here. */

error = SsslLibOpen( libRef );
...

SSL Library Structures and Data Types
The SSL library defines a number of structures and data types:

- SsslAttribute
- SsslContext
- SsslLib
- SsslCallback
- SsslCipherSuiteInfo
- SsslExtendedItem

SSL Library Functions

- SsslClose
- SsslConsume
- SsslContextCreate
- SsslContextDestroy
- SsslContextGetLong
- SsslContextGetPtr
- SsslContextSetLong
- SsslContextSetPtr
- SsslFlush
- SsslLibClose
- SsslLibCreate
- SsslLibDestroy
- SsslLibOpen
- SsslLibGetLong
- SsslLibGetPtr
- SsslLibSetLong
- SsslLibSetPtr
- SsslOpen
- SsslPeek
- SsslRead
- SsslReceive
- SsslSend
- SsslReceive
- SsslWrite

In addition, the SSL library defines one callback function:

- SsslCallbackFunc

SSL Library Attributes and Macros
You interact with SSL attributes using the following macros:

- SsslContextGet_Attribute (integer version)
- SsslContextGet_Attribute (pointer version)
- SsslLibSet_Attribute (integer version)
- SsslContextSet_Attribute (pointer version)
• `SslLibGet_Attribute (integer version)`
• `SslLibGet_Attribute (pointer version)`
• `SslLibSet_Attribute (integer version)`
• `SslLibSet_Attribute (pointer version)`

The attributes that you can manipulate with the above macros are:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppInt32</td>
<td>LastIO</td>
</tr>
<tr>
<td>AppPtr</td>
<td>Mode</td>
</tr>
<tr>
<td>AutoFlush</td>
<td>PeerCert</td>
</tr>
<tr>
<td>BufferedReuse</td>
<td>PeerCommonName</td>
</tr>
<tr>
<td>CipherSuite</td>
<td>ProtocolVersion</td>
</tr>
<tr>
<td>CipherSuiteInfo</td>
<td>RbufSize</td>
</tr>
<tr>
<td>CipherSuites</td>
<td>ReadBufPending</td>
</tr>
<tr>
<td>Compat</td>
<td>ReadOutstanding</td>
</tr>
<tr>
<td>DontSendShutdown</td>
<td>ReadRecPending</td>
</tr>
<tr>
<td>DontWaitForShutdown</td>
<td>ReadStreaming</td>
</tr>
<tr>
<td>Error</td>
<td>SessionReused</td>
</tr>
<tr>
<td>HsState</td>
<td>Socket</td>
</tr>
<tr>
<td>InfoCallback</td>
<td>SslSession</td>
</tr>
<tr>
<td>InfoInterest</td>
<td>SslVerify</td>
</tr>
<tr>
<td>IoTimeout</td>
<td>Streaming</td>
</tr>
<tr>
<td>IoFlags</td>
<td>VerifyCallback</td>
</tr>
<tr>
<td>IoStruct</td>
<td>WbufSize</td>
</tr>
<tr>
<td>LastAlert</td>
<td>WriteBufPending</td>
</tr>
<tr>
<td>LastApi</td>
<td></td>
</tr>
</tbody>
</table>
1.0 Float Manager

This section provides reference material for the Palm OS 1.0 float manager. In Palm OS 1.0, the float manager API is declared in the header file `FloatMgr.h`. In Palm OS 2.0, this file was renamed to `FloatMgrOld.h`. In Palm OS 3.5, this file was made private.

**NOTE:** New applications should no longer use the 1.0 Float Manager. See Chapter 32, “Float Manager,” on page 695 for the functions now provided by the Float Manager.

### Float Manager Functions

**FplAdd**

**Purpose**
Add two floating-point numbers (returns \( a + b \)).

**Prototype**

```c
FloatType FplAdd (FloatType a, FloatType b)
```

**Parameters**

\( a, b \)  
The floating-point numbers.

**Result**

Returns the normalized floating-point result of the addition.

**Comments**

Under Palm OS® 2.0 and later, most applications will want to use the arithmetic symbols instead. See the “Floating-Point” section in the *Palm OS Programmer’s Companion*, vol. I.

**Compatibility**

This function is not supported by PACE ([5.0 New Feature Set](#)).
1.0 Float Manager

Float Manager Functions

FplAToF

Purpose
Convert a zero-terminated ASCII string to a floating-point number. The string must be in the format: [-]x.[yyyyyyyy][e[-]zz]

Prototype
FloatType FplAToF (char* s)

Parameters
s Pointer to the ASCII string.

Result
Returns the floating-point number.

Comments
The mantissa of the number is limited to 32 bits.

Compatibility
This function is not supported by PACE (5.0 New Feature Set).

See Also
FplFToA, FplFree, FplInit

FplBase10Info

Purpose
Extract detailed information on the base 10 form of a floating-point number: the base 10 mantissa, exponent, and sign.

Prototype
Err FplBase10Info (FloatType a, ULong* mantissaP, Int* exponentP, Int* signP)

Parameters
a The floating-point number.
mantissaP The base 10 mantissa (return value).
exponentP The base 10 exponent (return value).
signP The sign, 1 or -1 (return value).

Result
Returns an error code, or 0 if no error.

Comments
The mantissa is normalized so it contains at least kMaxSignificantDigits significant digits when printed as an integer value.
FlpBaseInfo reports that zero is "negative"; that is, it returns a one for xSign. If this is a problem, a simple workaround is:

```c
if (xMantissa == 0) {
    xSign = 0;
}
```

**Compatibility**

This function is not supported by PACE (5.0 New Feature Set).

**FplDiv**

**Purpose**

Divide two floating-point numbers (result = dividend/divisor).

**Prototype**

```c
FloatType FplDiv (FloatType dividend, FloatType divisor)
```

**Parameters**

- `dividend` Floating-point dividend.
- `divisor` Floating-point divisor.

**Result**

Returns the normalized floating-point result of the division.

Under Palm OS 2.0 and later, most applications will want to use the arithmetic symbols instead. See the “Floating-Point” section in the Palm OS Programmer’s Companion, vol. I.

**Compatibility**

This function is not supported by PACE (5.0 New Feature Set).

**FplFloatToLong**

**Purpose**

Convert a floating-point number to a long integer.

**Prototype**

```c
Long FplFloatToLong (FloatType f)
```

**Parameters**

- `f` Floating-point number to be converted.

**Result**

Returns the long integer.
1.0 Float Manager
Float Manager Functions

Compatibility
This function is not supported by PACE (5.0 New Feature Set).

See Also FplLongToFloat, FplFloatToULong

FplFloatToULong

Purpose
Convert a floating-point number to an unsigned long integer.

Prototype
ULong FplFloatToULong (FloatType f)

Parameters
f
Floating-point number to be converted.

Result
Returns an unsigned long integer.

Compatibility
This function is not supported by PACE (5.0 New Feature Set).

See Also FplLongToFloat, FplFloatToLong

FplFree

Purpose
Release all memory allocated by the floating-point initialization.

Prototype
void FplFree()

Parameters
None.

Result
Returns nothing.

Comments
Applications must call this routine after they’ve called other functions that are part of the float manager.

Compatibility
If 5.0 New Feature Set is present, this function acts as a NOP.

See Also FplInit
FplFToA

Purpose
Convert a floating-point number to a zero-terminated ASCII string in exponential format: \([-]x.yyyyyye[-]zz\)

Prototype
Err FplFToA (FloatType a, char* s)

Parameters
a
Floating-point number.
s
Pointer to buffer to contain the ASCII string.

Result
Returns an error code, or 0 if no error.

Compatibility
This function is not supported by PACE (5.0 New Feature Set).

See Also
FplAToF, FplFree, FplInit

FplInit

Purpose
Initialize the floating-point conversion routines.
Allocate space in the system heap for floating-point globals.
Initialize the tenPowers array in the globals area to the powers of 10 from -99 to +99 in floating-point format.

Prototype
Err FplInit()

Parameters
None.

Result
Returns an error code, or 0 if no error.

Comments
Applications must call this routine before calling any other Fpl function.

Compatibility
If 5.0 New Feature Set is present, this function acts as a NOP.

See Also
FplFree
1.0 Float Manager
Float Manager Functions

FplLongToFloat

Purpose  Convert a long integer to a floating-point number.

Prototype  FloatType FplLongToFloat (Long x)

Parameters  x  A long integer.

Result  Returns the floating-point number.

Compatibility  This function is not supported by PACE (5.0 New Feature Set).

FplMul

Purpose  Multiply two floating-point numbers.

Prototype  FloatType FplMul (FloatType a, FloatType b)

Parameters  a, b  The floating-point numbers.

Result  Returns the normalized floating-point result of the multiplication.

Comments  Under Palm OS 2.0 and later, most applications will want to use the arithmetic symbols instead. See the “Floating-Point” section in the Palm OS Programmer’s Companion, vol. I.

Compatibility  This function is not supported by PACE (5.0 New Feature Set).

FplSub

Purpose  Subtract two floating-point numbers (returns a - b).

Prototype  FloatType FplSub (FloatType a, FloatType b)

Parameters  a, b  The floating-point numbers.

Result  Returns the normalized floating-point result of the subtraction.
<table>
<thead>
<tr>
<th><strong>Comments</strong></th>
<th>Under Palm OS 2.0 and later, most applications will want to use the arithmetic symbols instead. See the “Floating-Point” section in the <em>Palm OS Programmer’s Companion</em>, vol. I.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compatibility</strong></td>
<td>This function is not supported by PACE (<em>5.0 New Feature Set</em>).</td>
</tr>
</tbody>
</table>
1.0 Float Manager

Float Manager Functions
Index

Symbols
_searchF 967

Numerics
2.0 feature set 2304
3.0 feature set 2308
3.1 feature set 2312
3.2 feature set 2315
3.5 feature set 2324, 2330, 2342, 2346, 2351

A
accented characters and StrToLower 936
active form 284, 285
active window 68, 1170, 1230
adding event to event queue 942
AddrLookupParamsType 494
AlarmMgr.h 505
alarms 505–509
  and launch codes 17
  canceling 507
  procedure alarms 508
  setting 507
  sysAppLaunchCmdTimeChange 35
alerts 275
  custom alert 278, 279
  SysFatalAlert 418
allocating chunks on dynamic heap 794
AlmAlarmProcPtr 509
almErrFull 507, 508
almErrMemory 507, 508
AlmGetAlarm 505
AlmGetProcAlarm 506
almProcCmdCustom 510
AlmProcCmdEnum 509
almProcCmdReschedule 510
almProcCmdTriggered 510
AlmSetAlarm 507
AlmSetProcAlarm 508
alphaGraffitiSilkscreenArea 954
APAlgorithmEnum 2085
APKeyClassEnum 2089
APKeyUsageEnum 2089
APModeEnum 2084

APPaddingEnum 2090
appErrorClass 645
appEvtHookKeyMask 57
AppInfoPtr 133
appInfoStringsRsc 146
AppInt32 SSL Attribute 2187
AppLaunchCmd.h 493
application preferences 842
applications
  Security 34
AppPtr SSL Attribute 2188
APProviderContextStruct 2095
APProviderInfoStruct 2095
APProviderInfoType 2095
appStopEvent 43
archiving
  marking record as archived 571
ARM-native code
  calling 1258
atoi function substitute (StrAToI) 919
Attention Manager 91, 105
AttnCallbackProc 130
AttnCommandArgsType 107
AttnCommandType 105
AttnDoSpecialEffects 119
attnErrMemory 116
AttnFlagsType 111
AttnForgetIt 120
AttnGetAttention 121
AttnGetCounts 124
AttnIndicatorEnable 125
AttnIndicatorEnabled 126
AttnIterate 126
AttnLaunchCodeArgsType 113
AttnLevelType 114
AttnListOpen 127
AttnNotifyDetailsType 91
AttnUpdate 128
AutoFlush SSL Attribute 2188
auto-off
  setting 990
  timer 957
autoRepeatKeyMask 57
<table>
<thead>
<tr>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>badDrawWindowValue 1236</td>
</tr>
<tr>
<td>BarBeamBitmap 375</td>
</tr>
<tr>
<td>BarCopyBitmap 374</td>
</tr>
<tr>
<td>BarCutBitmap 374</td>
</tr>
<tr>
<td>BarDeleteBitmap 374</td>
</tr>
<tr>
<td>BarInfoBitmap 375</td>
</tr>
<tr>
<td>BarPasteBitmap 374</td>
</tr>
<tr>
<td>BarSecureBitmap 374</td>
</tr>
<tr>
<td>BarUndoBitmap 374</td>
</tr>
<tr>
<td>base 10 form of floating-point number 698, 2356</td>
</tr>
<tr>
<td>battery timeout 963, 965</td>
</tr>
<tr>
<td>battery voltage warning threshold 963, 965</td>
</tr>
<tr>
<td>Bitmap.h 513</td>
</tr>
<tr>
<td>BitmapCompressionType 513</td>
</tr>
<tr>
<td>BitmapDirectInfoType 514</td>
</tr>
<tr>
<td>BitmapFlagsType 515</td>
</tr>
<tr>
<td>BitmapPtr 518</td>
</tr>
<tr>
<td>bitmapRsc 535</td>
</tr>
<tr>
<td>bitmaps</td>
</tr>
<tr>
<td>drawing 1172</td>
</tr>
<tr>
<td>BitmapType 518</td>
</tr>
<tr>
<td>BitmapTypeV0 521</td>
</tr>
<tr>
<td>BitmapTypeV1 522</td>
</tr>
<tr>
<td>BitmapTypeV2 524</td>
</tr>
<tr>
<td>BitmapTypeV3 527</td>
</tr>
<tr>
<td>BitmapVersionOne 535</td>
</tr>
<tr>
<td>BitmapVersionTwo 535</td>
</tr>
<tr>
<td>BitmapVersionZero 535</td>
</tr>
<tr>
<td>blank lines in field 213</td>
</tr>
<tr>
<td>BmpBitsSize 536</td>
</tr>
<tr>
<td>BmpColortableSize 537</td>
</tr>
<tr>
<td>BmpCompress 537</td>
</tr>
<tr>
<td>BmpCreate 538</td>
</tr>
<tr>
<td>BmpCreateBitmapV3 541</td>
</tr>
<tr>
<td>BmpDelete 542</td>
</tr>
<tr>
<td>BmpGetBits 543</td>
</tr>
<tr>
<td>BmpGetColortable 544</td>
</tr>
<tr>
<td>BmpGetCompressionType 544</td>
</tr>
<tr>
<td>BmpGetDensity 545</td>
</tr>
<tr>
<td>BmpGetNextBitmapAnyDensity 548</td>
</tr>
<tr>
<td>BmpGetTransparentValue 550</td>
</tr>
<tr>
<td>BmpGetVersion 551</td>
</tr>
</tbody>
</table>

| BmpGlueGetBitDepth 544, 1892 |
| BmpGlueGetCompressionType 1895 |
| BmpGlueGetDimensions 546, 1892 |
| BmpGlueGetNextBitmap 547, 1892 |
| BmpGlueGetTransparentValue 1895 |
| BmpGlueSetTransparentValue 1896 |
| BmpSetDensity 551 |
| BmpSetTransparentValue 552 |
| BmpSize 553 |
| boldFont 711, 736, 1903 |
| boot, and heap compacting 784 |
| bound of next line for global find 250 |
| BtLibAccessibleModeEnum 1940 |
| btLibActiveMode 1953 |
| BtLibAddrAToBtd 1930 |
| BtLibAddrBtdToA 1931 |
| btLibCachedOnly 1972 |
| btLibCachedThenRemote 1972 |
| BtLibCancelInquiry 1962 |
| BtLibClassOfDeviceType 1941 |
| BtLibClose 1960 |
| btLibConnectableOnly 1941 |
| BtLibConnectionRoleEnum 1946 |
| BtLibDeviceAddressType 1946 |
| btLibDiscoverableAndConnectable 1941 |
| BtLibDiscoverMultipleDevices 1963 |
| BtLibDiscoverSingleDevice 1966 |
| BtLibFriendlyNameType 1947 |
| BtLibGeneralPreferenceEnum 1969 |
| BtLibGetGeneralPreference 1968 |
| BtLibGetNameEnum 1971 |
| BtLibGetRemoteDeviceName 1970 |
| BtLibGetSelectedDevices 1972 |
| BtLibHandleEvent 2297 |
| BtLibHandleTransportEvent 2297 |
| btLibHoldMode 1953 |
| BtLibL2CapHToNL 1932 |
| BtLibL2CapHToNS 1932 |
| BtLibL2CapNToHL 1933 |
| BtLibL2CapNToHS 1933 |
| BtLibL2CapPsmType 1992 |
| BtLibLanguageBaseTripletType 1992 |
| BtLibLinkConnect 1973 |
BtLibLinkDisconnect 1974
BtLibLinkGetState 1976
BtLibLinkModeEnum 1953
btLibLinkPrefAuthenticated 1979
btLibLinkPrefEncrypted 1979
btLibLinkPrefLinkRole 1979
BtLibLinkPrefsEnum 1978
BtLibLinksetState 1977
BtLibManagementCallback 1990
btLibManagementEventAccessibilityChange 1949
btLibManagementEventAclConnectInbound 1949
btLibManagementEventAclConnectOutbound 1949
btLibManagementEventAclDisconnect 1950
btLibManagementEventAuthenticationComplete 1950
btLibManagementEventEncryptionChange 1950
btLibManagementEventInquiryCanceled 1951
btLibManagementEventInquiryComplete 1951
btLibManagementEventInquiryResult 1951
btLibManagementEventLocalNameChange 1951
btLibManagementEventModeChange 1952
btLibManagementEventNameResult 1953
btLibManagementEventPasskeyRequested 1954
btLibManagementEventPiconetCreated 1954
btLibManagementEventPiconetDestroyed 1954
btLibManagementEventRadioState 1955
btLibManagementEventRoleChange 1956
BtLibManagementEventType 1947
btLibMasterRole 1946
BtLibMutex 2297
btLibNotAccessible 1941
BtLibOpen 1960
BtLibOpenBackground 2297
btLibParkMode 1953
BtLibPiconetCreate 1979
BtLibPiconetDestroy 1981
BtLibPiconetLockInbound 1982
BtLibPiconetUnlockInbound 1983
btLibPrefCurrentAccessible 1969
btLibPrefLocalClassOfDevice 1969
btLibPrefName 1969
btLibPrefUnconnectedAccessible 1970
BtLibProfileDescriptorListEntryType 2003
BtLibProtocolDescriptorListEntryType 2004
BtLibRegisterManagementNotification 1984
btLibRemoteOnly 1972
BtLibRfCommHtoNL 1934
BtLibRfCommHtoNS 1934
BtLibRfCommNtoHL 1935
BtLibRfCommNtoHS 1935
BtLibRfCommServerIdType 2004
BtLibSdpAttributeDataType 2005
BtLibSdpAttributeIdType 2006
BtLibSdpCompareUuuids 2042
BtLibSdpGetPSMByUuid 2043
BtLibSdpGetRawDataElementType 2044
BtLibSdpGetRawDataElementSize 2044
BtLibSdpGetServerChannelByUuid 2047
BtLibSdpHtoNL 1936
BtLibSdpHtoNS 1936
BtLibSdpNtoHL 1937
BtLibSdpNtoHS 1937
BtLibSdpParseRawDataElement 2048
BtLibSdpRecordHandle 2007
BtLibSdpRemoteServiceRecordHandle 2007
BtLibSdpServiceRecordCreate 2050
BtLibSdpServiceRecordDestroy 2051
BtLibSdpServiceRecordGetAttribute 2052
BtLibSdpServiceRecordGetNumListEntries 2054
BtLibSdpServiceRecordGetNumLists 2056
BtLibSdpServiceRecordGetRawAttribute 2058
BtLibSdpServiceRecordGetSizeOfRawAttribute 2060
BtLibSdpServiceRecordGetStringOrURLLength 2062
BtLibSdpServiceRecordMapRemote 2064
BtLibSdpServiceRecordSetActiveAttribute 2065
BtLibSdpServiceRecordSetActiveAttributesForSocket 2067
BtLibSdpServiceRecordSetRawAttribute 2068
BtLibSdpServiceRecordsGetByServiceClass 2070
BtLibSdpServiceRecordStartAdvertising 2072
BtLibSdpServiceRecordStopAdvertising 2073
BtLibSdpUuidInitialize 2074
BtLibSdpUuidSizeEnum 2007
BtLibSdpUuidType 2008
BtLibSdpVerifyRawDataElement 2075
BtLibSecurityFindTrustedDeviceRecord 1926
BtLibSecurityGetTrustedDeviceRecordInfo 1927
BtLibSecurityNumTrustedDeviceRecords 1928
BtLibSecurityRemoveTrustedDeviceRecord 1929
BtLibServiceClose 2297
BtLibServiceIndicateSessionStart 2297
BtLibServiceOpen 2297
BtLibServicePlaySound 2297
BtLibSetGeneralPreference 1985
btLibSlaveRole 1946
BtLibSleep 2297
btLibSniffMode 1953
BtLibSocketAdvanceCredit 2024
BtLibSocketCallback 2076
BtLibSocketClose 2025
BtLibSocketConnect 2026
BtLibSocketConnectInfoType 2027
BtLibSocketCreate 2029
btLibSocketEventConnectedInbound 2012
btLibSocketEventConnectedOutbound 2012
btLibSocketEventConnectRequest 2013
btLibSocketEventData 2013
btLibSocketEventDisconnected 2014
btLibSocketEventSdpGetAttribute 2014
btLibSocketEventSdpGetNumListEntries 2017
btLibSocketEventSdpGetNumLists 2017
btLibSocketEventSdpGetPsmByUuid 2021
btLibSocketEventSdpGetRawAttribute 2018
btLibSocketEventSdpGetRawAttributeSize 2019
btLibSocketEventSdpGetServerChannelByUuid 2020
btLibSocketEventSdpGetStringLen 2016
btLibSocketEventSdpServiceRecordHandle 2014
btLibSocketEventSendComplete 2022
BtLibSocketEventType 2010
BtLibSocketGetInfo 2031
btLibSocketInfo_L2CapChannel 2033
btLibSocketInfo_L2CapPsm 2033
btLibSocketInfo_MaxRxSize 2033
btLibSocketInfo_MaxTxSize 2033
btLibSocketInfo_Protocol 2033
btLibSocketInfo_RemoteDeviceAddress 2033
btLibSocketInfo_RfCommOutstandingCredits 2034
btLibSocketInfo_RfCommServerId 2034
btLibSocketInfo_SdpServiceRecordHandle 2034
btLibSocketInfo_SendPending 2034
btLibSocketInfoEnum 2032
BtLibSocketListen 2034
BtLibSocketListenInfoType 2036
BtLibSocketRef 2010
BtLibSocketRespondToConnection 2038
BtLibSocketSend 2040
BtLibStartInquiry 1987
BtLibStringType 2011
BtLibUnload 2297
BtLibUnregisterManagementNotification 1988
BtLibUrlType 2011
BtLibWake 2297
BufferedReuse SSL Attribute 2189
busy bit 619
ButtonFrameType 157
buttons (silk-screened buttons) 56
byteAttrFirst 998
byteAttrLast 998
byteAttrMiddle 998
byteAttrSingle 998

calibrating the pen 825
canceling alarms 507
capsLockMask 57
card number 777
categories, setting label 313
category
    DmSeekRecordInCategory 631
    moving records 603
Category Constants 133
Category Data Structures 133
Category Functions 133
CategoryCreateList 136, 142, 2306
CategoryCreateListV10 138
categoryDefaultEditCategoryString 135, 137, 139, 147, 2329
categoryDefaultEditString 135, 137, 148
categoryEdit 139, 2306
CategoryEditV10 141
CategoryEditV2 140
CategoryFind 141
CategoryFreeList 142, 2306
CategoryFreeListV10 143
CategoryGetName 144
CategoryGetNext 145
categoryHideEditCategory 135, 137, 148, 2329
CategoryInitialize 134, 146
CategorySelect 147, 2306
CategorySelectV10 149
CategorySetName 150
CategorySetTriggerLabel 151
CategoryTruncateName 152
cCountryName constants 767
character attribute functions 555–559
character encodings 1003, 1021, 1028, 1035
characters
  See Also multi-byte characters
  attributes 1000, 1004, 1005, 1006, 1008, 1009,
    1010, 1011, 1013
  converting 1036
  drawable 1011
  drawing in window 1174
  erasing 1181
  graphic 1006
  inverting 1198
  printable 1009
  size 1012
  sorting text 559
  valid 1011
CharAttr.h 555
charAttrAlNum 1000
charAttrAlpha 1000
charAttrCntrl 1000
charAttrDelim 1001
charAttrGraph 1001
charAttrPrint 1000
charAttrSpace 1000
charEncoding..., constants 766
CharEncodingType 765, 997
charEncodingUnknown 1029
checkboxFont 717
checkboxTableItem 424
ChrHorizEllipsis 555, 2313
ChrIsHardKey 556
ChrNumericSpace 556, 2313
chunks
  card number 777
  disposing of chunk 778
  heap ID 779, 793
  locking 779
  size 782
  unlocking 783, 797
CipherSuite SSL Attribute 2190
CipherSuiteInfo SSL Attribute 2190
CipherSuites SSL Attribute 2191
clipboard 205, 206, 226
Clipboard.h 153
ClipboardAddItem 154
ClipboardAppendItem 155
ClipboardFormatType 153
ClipboardGetItem 156
Clipper application 2318
clipping rectangle 1233
closing net library 1424, 1438
CncAddProfile 1273
CncDefineParamID 1275
CncDefineParamId 1272
CncDeleteProfile 1277
CncGetParamType 1272, 1278
CncGetProfileInfo 1278
CncGetProfileList 1280
CncGetSystemFlagBitnum 1283
CncGetTrueParamID 1283
CncIsFixedLengthParamType 1284
CncIsSystemFlags 1284
CncIsSystemRange 1285
CncIsThirdPartiesRange 1285
CncIsVariableLengthParamType 1286
cncNotifyProfileEvent 75
CncProfileCloseDB 1286
CncProfileCount 1287
CncProfileCreate 1275, 1288
CncProfileDelete 1288
CncProfileGetIDFromIndex 1263, 1290
CncProfileGetIDFromName 1263, 1291
CncProfileGetIndex 1291
CncProfileID 1263
date 188
date system resource 185
DateAdjust 1055
DateDaysToDate 1056
DateGlueTemplateToAscii 1060, 1892
DateGlueToDOWDMFormat 1064, 1892
DatePtr 1049
DateSecondsToDate 1056
dateStringLength 1061
dateStringLength 1061
dateTableItem 424
DateTemplateToAscii 1057
DateTime.h 1045
DateTimePtr 1048
DateTimeType 1048
DateTimeToAscii 1061
dateTableItem 424
Day.h 185
DayDrawDays 186
DayDrawDaySelector 187
DayHandleEvent 187
DayOfMonth 1051, 1064
DayOfWeek 1065
daySelectEvent 46
daySelectEvent 46
DaysInMonth 1066
deleting databases See Also DmDatabaseProtect
deleting records 584
defaultAlarmSoundLevel 840
defaultAlarmSoundVolume 840
defaultAutoLockTime 840
defaultAutoLockTimeFlag 840
defaultAutoUnlockSecs 840
defaultBoldFont 1903
defaultGameSoundLevel 840
defaultGameSoundVolume 840
defaultLargeFont 1902
defaultSmallFont 1902
defaultSysSoundLevel 840
defaultSysSoundVolume 840
defaultSystemFont 1902
delete bit 582, 585
delete callback function 1352
DeleteProc 1315, 1317, 1352
deleting databases See Also DmDatabaseProtect
deleting records 584
DensityType 531
DeviceInfoType 1572
DeviceInfoType structure 1551
dialogs
   Edit Categories 139
digitizer
      and PenResetCalibration function 826
      and penUpEvent 63
      EvtProcessSoftKeyStroke 956
DirectionType 229
DlkCallAppReplyParamType 1249
DlkControl 1248
DlkGetSyncInfo 1251
DLServer.h 1247
dmAllCategories 142, 563, 611
dmAllHdrAttrs 565
dmAllRecAttrs 563
DmArchiveRecord 571
DmAttachRecord 572
DmAttachResource 573
dmCategoryLength 133, 144, 563
DmCloseDatabase 574
DmComparF 590, 602, 621, 640
DmCreateDatabase 575
DmCreateDatabaseFromImage 577
DmDatabaseInfo 577
DmDatabaseProtect 574, 580
DmDatabaseSize 581
dmDBNameLength 570, 575, 633
DmDeleteCategory 582
DmDeleteDatabase 574, 583
DmDeleteRecord 584
DmDetachRecord 585
DmDetachResource 586
dmErrDatabaseNotProtected 567
dmErrRecordArchived 568
DmFindDatabase 576, 584, 587
DmFindRecordByID 587
DmFindResource 588
DmFindResourceType 589
DmFindSortPosition 590, 2306
DmFindSortPositionV10 591
DmGet1Resource 601, 602, 609
DmGetAppInfoID 592
DmGetDatabase 584, 593
DmGetDatabaseLockState 593
DmGetLastError 594, 595
DmGetNextDatabaseByTypeCreator 596
DmGetRecord 599
DmGetResource 600, 601
DmGetResourceIndex 601
dmHdrAttrAppInfoDirty 565
dmHdrAttrBackup 565
dmHdrAttrBundle 565
dmHdrAttrCopyPrevention 565
dmHdrAttrHidden 565
dmHdrAttrLaunchableData 566
dmHdrAttrOKToInstallNewer 566
dmHdrAttrOpen 566
dmHdrAttrReadOnly 566
dmHdrAttrRead Only 566
dmHdrAttrRecyclable 574
dmHdrAttrResDB 566
dmHdrAttrResetAfterInstall 566
dmHdrAttrStream 566
DmInsertionSort 602
dmMaxRecordIndex 563, 572, 607
dmModeExclusive 570
dmModeLeaveOpen 570
dmModeReadOnly 570
dmModeReadWrite 570
dmModeShowSecret 570
dmModeWrite 570
DmMoveCategory 603
DmMoveRecord 605
DmNewHandle 606
DmNewRecord 607
DmNewResource 608
DmNextOpenDatabase 609
DmNextOpenResDatabase 609
DmNumDatabases 610
DmNumRecords 611
DmNumRecordsInCategory 611
DmNumResources 612
DmOpenDatabase 570, 613
DmOpenDatabaseByTypeCreator 615
DmOpenDatabaseInfo 616
DmOpenDBNoOverlay 570, 617
DmOpenRef 561
DmPositionInCategory 618
DmQueryNextInCategory 619
DmQuickSort 621
dmRecAttrBusy 563
dmRecAttrCategoryMask 563, 604
dmRecAttrDelete 563
dmRecAttrDirty 563
dmRecAttrSecret 563
dmRecNumCategories 133, 563
DmRecordInfo 622
DmReleaseRecord 599, 607, 623
DmReleaseResource 608, 623
DmRemoveRecord 624
DmRemoveResource 625
DmRemoveSecretRecords 626
DmResID 561
DmResizeRecord 626
DmResizeResource 627
DmResourceInfo 628
DmResType 562
DmSearchRecord 629
DmSearchResource 601, 629
DmSearchStatePtr 596
DmSearchStateType 596
dmSeekBackward 631
dmSeekForward 631
DmSeekRecordInCategory 631
DmSet 632
DmSetDatabaseInfo 633
DmSetRecordInfo 635
DmSetResourceInfo 636
DmStrCopy 637
dmSysOnlyHdrAttrs 566
dmSysOnlyRecAttrs 563
dmUnfiledCategory 563, 1323
DmWrite 638
DmWriteCheck 639
DontSendShutdown SSL Attribute 2193
DontWaitForShutdown SSL Attribute 2193
doubleTapKeyMask 57
dowDateStringLength 1063
dowLongDateStringLength 1063
doze mode
    SysTaskDelay 992
Dragonball EZ 2313
draw window 1235
drawable characters 1011
drawDetail structure 108
drawing rectangular frame 1176, 1179, 1208
drawItemsCallback 343, 355
drawList structure 109
DrvStateType 1147
DrvEntryPoint
    for virtual driver 1538
DrvvrInfoType structure 1523
DrvvrRecvQType structure 1525
DrvvrStatusEnum 1526
dynamic heap
    adding chunk 780
    allocating chunk 794
    moving memory 790
    reinitializing 989
    test 784
dynamic heap handles 778
dynamic scrolling 66

e
Edit Categories dialog 139
editingStrID 137, 147
enabling windows 1171
erasing characters 1181
erasing lines in window 1182
erasing rectangle 1183
ErrAlert 644
ErrCatch 645, 650
ErrDisplay 646
ErrDisplayFileLineMsg 647
ErrEndCatch 647, 650
ErrExceptionList 648
ErrExceptionType 644
ErrFatalDisplayIf 648
errNone 1322, 1585
ErrNonFatalDisplayIf 649
error code from data manager call 594
error manager 643–650
Error SSL Attribute 2194
ERROR_CHECK_FULL 643
ERROR_CHECK_LEVEL 643, 646, 649
ERROR_CHECK_NONE 643
ERROR_CHECK_PARTIAL 643
ErrorBase.h 643
ErrorMgr.h 643
ErrThrow 650
ErrTry 650
event queue, adding event 942
Event.H 941
Event.h 39, 2328
EventPtr 43
events 39, 69
eventsEnum 40
EventType 39–69, 2327
EvtAddEventToQueue 942
EvtAddUniqueEventToQueue 942
EvtCopyEvent 944
EvtDequeuePenPoint 945
EvtDequeuePenStrokeInfo 945
EvtEnableGraffiti 946
EvtEnqueueKey 946
EvtEventAvail 947
EvtFlushKeyQueue 948
EvtFlushNextPenStroke 948
EvtFlushPenQueue 949
EvtGetEvent 62, 804, 805
EvtGetPen 950
EvtGetPenBtnList 951
EvtGetPenNative 952
EvtKeyQueueEmpty 955
EvtKeyQueueSize 955
EvtPenQueueSize 956
EvtProcessSoftKeyStroke 956
EvtResetAutoOffTimer 957
EvtSetAutoOffCmd 958
EvtSysEventAvail 959
evtWaitForever 62, 949, 1643, 1644
EvtWakeup 960, 1591
EvtWakeupWithoutNilEvent 960
exchange manager 1297, 2310
ExgAccept 21, 22, 1309, 1317, 1320, 1342
exgAskCancel 18
exgAskOK 18
ExgAskParamType 19
ExgAskResultType 1297
exgBeamPrefix 1308
exgBeamScheme 1308, 1309
ExgConnect 1310, 1320
ExgControl 1313
ExgDBDeleteProcPtr 659, 1122, 1315
ExgDBRead 1315, 1353
ExgDBReadProcPtr 1353
ExgDBWrite 1318
ExgDBWriteProcPtr 660, 1129, 1318
ExgDialogInfoType 1322, 1323
ExgDisconnect 19, 21, 1312, 1319, 1326, 1340, 1342,
1350
ExgDoDialog 18, 1298, 1322, 1335, 1337
exgErrNoKnownTarget 1327, 1332, 1333, 1335,
1337, 1338, 1351
exgErrNotSupported 1302, 1335
ExgGet 1312, 1320, 1325, 1326, 1342
exgGet 1337
ExgGetDefaultApplication 1327
ExgGetRegisteredApplications 1328
ExgGetRegisteredTypes 1330
ExgGetTargetApplication 1331, 1336, 1337
ExgGoToType 1298
ExgLibAccept 1357
exgLibAPIVersion 1314
ExgLibClose 1358
ExgLibConnect 1359
ExgLibControl 1360
exgLibCtlGetPreview 1314
exgLibCtlGetTitle 1314
exgLibCtlGetVersion 1314
exgLibCtlSpecificOp 1315
ExgLibDisconnect 1362
ExgLibGet 1363
ExgLibHandleEvent 1364
ExgLibOpen 1365
ExgLibPut 1366
ExgLibReceive 1368
ExgLibRequest 1369
ExgLibSend 1370
ExgLibSleep 1371
exgLibSmsIncompleteDeleteOp 2233
exgLibSmsIncompleteGetCountOp 2233
exgLibSmsPrefDisplayOp 2233
exgLibSmsPrefGetDefaultOp 2233
exgLibSmsPrefGetOp 2233
exgLibSmsPrefSetOp 2233
ExgLibWake 1372
ExgLocalLib.h 1297
exgLocalOpAccept 1300
exgLocalOpGet 1300
exgLocalOpGetSender 1300
exgLocalOpNone 1300
exgLocalOpPut 1300
ExgLocalOpType 1299
exgLocalPrefix 1305, 1309
exgLocalScheme 1308, 1309
ExgLocalSocketInfoType 1299
exgMaxDescriptionLength 1329, 1343
exgMaxTitleLen 1329
exgMaxTypeLength 1329, 1330, 1343
exgMemError 1331, 1344
ExgMgr.h 653, 1297
exgNoAsk 1337
ExgNotifyGoto 1320, 1321, 1334, 1339
ExgNotifyPreview 1301, 1323, 1335
ExgNotifyReceive 1333, 1334, 1336
exgPreviewDialog 20, 1301
exgPreviewDraw 20, 1301
exgPreviewFirstUser 21, 1302
ExgPreviewInfoType 1300, 1301, 1335
exgPreviewLastUser 21, 1302
exgPreviewLongString 20, 1301
exgPreviewQuery 20, 1302
exgPreviewShortString 1302
ExgPut 19, 1312, 1319, 1320, 1339, 1350
ExgReceive 21, 22, 1312, 1317, 1326, 1341, 1353
exgRegCreatorID 1307, 1345
exgRegExtensionID 1307, 1327
ExgRegisterData 28
ExgRegisterDatatype 1342
ExgRegisterDataV35 1347
exgRegSchemeID 1307, 1328, 1346
exgRegTypeID 1307, 1345
ExgRequest 27, 1348
ExgSend 19, 1312, 1319, 1340, 1349, 1355
exgSendBeamPrefix 1309
exgSendPrefix 1308
exgSendScheme 1308, 1309, 1346
ExgSetDefaultApplication 1345, 1346, 1350
ExgSocketType 19, 656, 1298, 1303, 1315, 1318
exgTitleBufferSize 1314
exgUnwrap 1337, 1338, 1344
Expansion Manager 79, 80, 653
expCapabilityHasStorage 654
expCapabilityReadOnly 654
expCapabilitySerial 654
ExpCardGetSerialPort 656
ExpCardInfo 657
ExpCardInfoType 653
ExpCardPresent 658
expErrCardBadSector 655
expErrCardNoSectorReadWrite 655
expErrCardNotPresent 654
expErrCardProtectedSector 655
expErrCardReadOnly 655
expErrEnumerationEmpty 655
expErrIncompatibleAPIVer 655
expErrInvalidSlotRefNum 655
expErrNotEnoughPower 654
expErrNotOpen 655
expErrSlotDeallocated 655
expErrStillOpen 655
expErrUnimplemented 655
expErrUnsupportedOperation 654
expHandledSound 80
expHandledVolume 80
expMediaType_Any 656
expMediaType_MacSim 656
expMediaType_MemoryStick 656
expMediaType_MultiMediaCard 656
expMediaType_PoserHost 656
expMediaType_RAMDisk 656
expMediaType_SecureDigital 656
expMediaType_SmartMedia 656
ExpSlotDriverInstall 659
ExpSlotDriverRemove 660
ExpSlotEnumerate 661
ExpSlotLibFind 662
extended font resource (nfnt) 721
extended gadget 259, 318, 327
EZ Dragonball 2313

F
fatal alert 418
FatalAlert.h 417
fcntl 1501
FeatureMgr.h 665
features See functions starting with Ftr
fgetc 899
fgets 900
field objects
  and text height 218
  modifying 208
  structure 198
Field.h 195
FieldAttrType 195
FieldPtr 198
FieldType 198
file mode constants 673, 674
file streaming 2309
FileClearerr 675
FileClose 675
FileControl 676
FileDelete 680
FileDmRead 681
FileEOF 682
FileError 683
FileFlush 683
FileGetLastError 684
FileInfoType 1075
FileOpen 685
FileOpEnum 677
FileOriginEnum 689
FileRead 687
FileRef 1076
FileRewind 688
FileSeek 688
FileStream.h 673
FileTell 689
FileTruncate 690
FileStream.h 673
FileDialogHeader 25, 249
Find (global find) 22, 26, 249–253
Find (lookup) 31
Find icon 56
Find.h 249
FindDrawHeader 25, 249
FindGetLineBounds 250
FindParamsType 22, 23
FindSaveMatch 24, 27, 250
FindStrInStr 24, 252
flags, launch flags 36
FldCalcFieldHeight 204
fldChangedEvent 47, 230, 2306
FldCompactText 205
FldCopy 205
FldCut 206
FldDelete 207
FldDirty 208
FldDrawField 208
fldEnterEvent 47, 221, 298
FldEraseField 209
FldFreeMemory 210
FldGetAttributes 211
FldGetBounds 211
FldSetFont 212
FldGetFont 212
FldGetInsPtPosition 212
FldGetMaxChars 213
FldGetNumberOfBlankLines 213
FldGetScrollPosition 214
FldGetScrollValues 214
FldGetSelection 215
FldGetTextAllocatedSize 216
FldGetTextHandle 217
FldGetTextHeight 218
FldGetTextLength 219
FldGetTextPtr 219
FldGetVisibleLines 220
FldGrabFocus 220, 446
FldHandleEvent 47, 48, 221
fldHeightChangedEvent 48, 223, 231, 298
FldInsert 222
FldMakeFullyVisible 223
FldNewField 224
FldPaste 226
FldRecalculateField 227
FldReleaseFocus 228
FldScrollable 228
FldScrollField 229
FldSendChangeNotification 230
FldSendHeightChangeNotification 231
FldSetAttributes 231
FldSetBounds 232
FldSetDirty 234
FldSetFont 235
FldSetInsertionPoint 235
FldSetInsPtPosition 236
FldSetMaxChars 237
FldSetMaxVisibleLines 237
FldSetScrollPosition 238
FldSetSelection 239
FldSetText 240
FldSetTextAllocatedSize 242
FldSetTextHandle 243
FldSetTextPtr 245
FldSetUsable 246
FldUndo 246
FldWordWrap 247
FloatMgr.h 695
FloatMgr.h (Palm OS 1.0) 2355
FloatMgrOld.h 2355
flushing pen queue 949
fntAppCustomBase 712
FntAverageCharWidth 713, 716, 723
FntBaseLine 714, 716, 723
FntCharHeight 713, 716, 724
FntCharsInWidth 724
FntCharsWidth 725
FntDefineFont 727
FntDescenderHeight 714, 716, 728
fntExtendedFormatMask 715, 717
FntSetFont 711, 728
FntSetFontPtr 729
FntGetScrollValues 729
FntGlueGetDefaultFontID 736, 1902
FntGlueWCharWidth 726, 733, 1892
FntGlueWidthToOffset 734, 1892
FntLineHeight 714, 716, 731
FntLineWidth 731
fntMissingChar 717
FntSetFont 711, 732
fntTabChrWidth 717
FntWCharWidth 710, 714, 716, 726, 732
FntWidthToOffset 725, 733
FntWordWrap 734
FntWordWrapReverseNLines 735
focus
and modal window 1201
FrmGetFocus 287
FrmSetFocus 316
font resource (NFNT) 718
font resource, extended (nfnt) 721
Font.h 709
FontCharInfoPtr 709
FontCharInfoType 709, 714, 716
FontDefaultType 1902
FontDensityType 710, 717
FontID 711, 727
FontPtr 712
fonts
and FldGetFont 212
FontSelect 736
FontSelect.h 709
FontType 712, 719
FontTypeV2 710, 714, 721
form objects
    FormType structure 271
functions 274–325
form, active 284, 285
Form.h 255
FormActiveStateType 310, 312
FormAttrType 255
FormBitmapType 257
FormCheckResponseFuncType 274, 279, 325
FormEventHandlerType 272, 327
FormFrameType 257
FormGadgetAttrType 258
formGadgetDeleteCmd 280, 283, 327
formGadgetDrawCmd 322, 328
formGadgetEraseCmd 301, 328
formGadgetHandleEventCmd 328
FormGadgetHandlerType 49, 50, 280, 283, 301, 322, 327
FormGadgetType 49, 51, 261, 327
FormGadgetTypeInCallback 262
FormLabelType 263
FormLineType 264
FormObjAttrType 265
FormObjectKind 265
FormObjectType 267
FormObjListType 268
FormPopupType 269
FormPtr 270
FormRectangleType 270
FormTitleType 271
FormType 271, 1906
FplAdd 2355
FplAToF 2356
FplBase10Info 2356
FplDiv 2357
FplFloatToLong 2357
FplFloatToULong 2358
FplFree 2358
FplFToA 2359
FplInit 2359
FplLongToFloa 2360
FplMul 2360
FplSub 2360
fprintf 900
fputc 901
fputs 902
frame type constants 1152
FrameBitsType 1151
frames
drawing in window 1176, 1179, 1208
FrameType 1152
FrmAlert 275
FrmCloseAllForms 49, 276
frmCloseEvent 49, 276, 296, 298
FrmCopyLabel 276
FrmCopyTitle 277
FrmCustomAlert 278
FrmCustomResponseAlert 279
FrmDeleteForm 280, 327
FrmDispatchEvent 281, 297
FrmDoDialog 281
FrmDrawForm 282, 299, 328
FrmEraseForm 54, 283
frmGadgetEnterEvent 49, 298, 329
frmGadgetMispEvent 50, 298, 329
FrmGetActiveField 284
FrmGetActiveForm 284
FrmGetActiveFormID 285
FrmGetControlGroupSelection 285
FrmGetControlValue 286
FrmGetFirstForm 287
FrmGetFocus 287
FrmGetFormBounds 288
FrmGetFormId 288
FrmGetFormPtr 289
FrmGetGadgetData 261, 289
FrmGetLabel 290
FrmGetNumberOfObjects 290
FrmGetObjectBounds 162, 163, 165, 291
FrmGetObjectId 291
FrmGetObjectIndex 292
FrmGetObjectPosition 294
FrmGetObjectPtr 294
FrmGetObjectType 295
FrmGetTitle 295
FrmGetWindowHandle 296
FrmGlueGetActiveField 1904
FrmGlueGetDefaultButtonID 1904
FrmGlueGetHelpID 1905
FrmGlueGetLabelFont 1905
FrmGlueGetMenuBarID 1906
FrmGlueSetDefaultButtonID 1907
FrmGlueSetHelpID 1907
FrmGlueSetLabelFont 1908
frmGotoEvent 51
FrmGotoForm 49, 52, 296
FrmGraffitiStateType 273
FrmHandleEvent 49, 53, 63, 297, 328
FrmHelp 300
FrmHideObject 169, 172, 259, 301, 328
Frm InitForm 302
frmInvalidObjectId 292
frmLoadEvent 52, 296
FrmNewBitmap 303
FrmNewForm 304
FrmNewGadget 305
FrmNewGsi 306
FrmNewLabel 307
frmNoSelectedControl 274, 285
frmOpenEvent 51, 52, 272, 281, 296, 309
FrmPointInTitle 308
FrmPopupForm 52, 309
FrmRedrawUpdateCode 55, 274, 322
FrmRemoveObject 309
FrmResponseCreate 274, 326
frmResponseQuit 274, 326
FrmRestoreActiveState 310, 312
FrmReturnToForm 311
FrmSaveActiveState 310, 312
FrmSaveAllForms 53, 312
frmSaveEvent 53, 312
FrmSetActiveForm 68, 313
FrmSetActiveForm 68, 313
FrmSetCategoryLabel 313
FrmSetControlGroupSelection 314
FrmSetControlValue 315
FrmSetEventHandler 316
FrmSetFocus 316, 446
FrmSetGadgetData 261, 317
FrmSetGadgetHandler 261, 318
FrmSetMenu 319, 379, 385, 386
FrmSetObjectBounds 162, 163, 165, 319

2376  Palm OS Programmer’s API Reference
FrmSetObjectPosition 320
FrmSetTitle 320
FrmShowObject 184, 259, 321, 328
frmTitleEnterEvent 53, 298
frmTitleSelectEvent 53, 298, 2328
FrmUpdateEvent 54, 274, 282, 299, 322
FrmUpdateForm 54, 55, 322
FrmUpdateScrollers 323
FrmValidatePtr 323
FrmVisible 325
ftrErrNoSuchFeature 665, 666, 667, 669, 670, 671
FtrGet 665
FtrGetByIndex 666
FtrPtrFree 667
FtrPtrNew 667
FtrPtrResize 669
FtrSet 670
FtrUnregister 670

G

gadget resource 259, 317
   extended 259, 318, 327
getchar 902
GetCharAttr 557
GetCharCaselessValue 558
GetCharSortValue 559
gethostname 1481
gets 903
GetSize 1546
GetSpace 1547
global find 22, 26, 249–253, 1022, 1914, 1918
   FindDrawHeader 249
   FindGetLineBounds 250
   saving data 33
gotIt structure 110
goto (global find) 26
GoToParamsType 26
Graffiti
   Command shortcuts 60
   enabling and disabling 946
Graffiti manager
   functions 737–747
Graffiti recognizer
   EvtDequeuPenPoint 945
   Graffiti Reference Dialog 418
   Graffiti Shift
      functions 331–333
   Graffiti.h 737
   GraffitiReference.h 417
   GraffitiShift.h 331
   GraffitiUI.h 417
   graphic characters 1006
   GraphicControlType 162, 167
   GraphicStatePtr 1159
   GrfAddMacro 737
   GrfAddPoint 738
   GrfCleanState 738
   GrfDeleteMacro 739
   GrfFilterPoints 739
   GrfFindBranch 740
   GrfFlushPoints 740
   GrfGetAndExpandMacro 741
   GrfGetGlyphMapping 741
   GrfGetMacro 742
   GrfGetMacroName 742
   GrfGetNumPoints 743
   GrfGetPoint 743
   GrfGetState 744
   GrfInitState 745
   GrfMatch 745
   GrfMatchGlyph 746
   GrfProcessStroke 746
   GrfSetState 747
   groups of controls 285
   GsiEnable 331
   GsiEnabled 332
   GsiInitialize 332
   GsiSetLocation 332
   GsiSetShiftState 333

H

hard reset 35
header line for global find 249
heap ID 786, 793
   of chunk 779
heaps
   compacting 784
free bytes 785
ROM based 785
height of text in field 218
Helper API 749
Helper.h 749
HelperNotifyEnumerateListType 749
HelperNotifyEventType 749, 751, 754
HelperNotifyExecuteType 752, 755, 756
HelperNotifyValidateType 754
HelperServiceClass.h 749
HelperServiceEMailDetailsType 754, 755
HelperServiceSMSDetailsType 754
HelperServiceSMSDetailType 756
HostControl.h 980
hostent 1414
HostGremlinIsRunning 980
HotSync and sysAppLaunchCmdSyncNotify 34
HotSync operation 101
HsState SSL Attribute 2194

icons 56
iconType 535
ID
databases 587
heap 786
iMessenger application 2318
IndexedColorType 1153
INetCacheEntryType 1856
INetCacheInfoType 1854
inetCfgName... constants 1850–1851
inetCompressionType... constants 1840
INetCompressionTypeEnum 1839
INetConfigNameType 1840
inetContentType... constants 1841
INetContentTypeEnum 1841
inetHTTPAttr... constants 1843–1844
INetHTTPAttrEnum 1841
INetLibCacheGetObject 1853
INetLibCacheList 1854
INetLibCheckAntennaState 1856
INetLibClose 1857
INetLibConfigAliasGet 1858
INetLibConfigAliasSet 1859
INetLibConfigDelete 1860
INetLibConfigIndexFromName 1861
INetLibConfigList 1862
INetLibConfigMakeActive 1863
INetLibConfigRename 1864
INetLibConfigSaveAs 1865
INetLibGetEvent 1866
INetLibOpen 1867
INetLibSettingGet 1868
INetLibSettingSet 1869
INetLibSockClose 1870
INetLibSockConnect 1871
INetLibSockHTTPAttrGet 1872
INetLibSockHTTPAttrSet 1873
INetLibSockHTTPRequestCreate 1874
INetLibSockHTTPRequestSend 1875
INetLibSockOpen 1877
INetLibSockRead 1878
INetLibSockSettingGet 1879
INetLibSockSettingSet 1880
INetLibSockStatus 1881
INetLibURLCrack 1882
INetLibURLGetInfo 1884
INetLibURLOpen 1885
INetLibURLsAdd 1886
INetLibWiCmd 1888
INetMgr.h 39, 1839, 2237
inetOpenURFlag... constants 1852
inetScheme... constants 1844–1845
INetSchemeEnum 1844
inetSetting... constants 1846–1847
INetSettingEnum 1845
inetSockReadyEvent 55
inetSockSetting... constants 1848–1849
INetSockSettingEnum 1847
inetSockStatusChangeEvent 56
inetStatus... constants 1850
INetStatusEnum 1849
INetURLInfo type 1884
INetURLInfoFlag... constants 1852
INetURLOpen 1883
InfoInterest SSL Attribute 2196
<table>
<thead>
<tr>
<th>Insertion point functions</th>
<th>Insertion points</th>
<th>Inverting characters in draw window</th>
<th>Inverting line in draw window</th>
</tr>
</thead>
<tbody>
<tr>
<td>335–338</td>
<td>and FlgGetInsPtPosition 212 and FlgGrabFocus 220 and FlgReleaseFocus 228 and FlgSetInsPos 235</td>
<td>1198</td>
<td>1199</td>
</tr>
<tr>
<td>insertion sort 983</td>
<td>displayed in field 209</td>
<td></td>
<td></td>
</tr>
<tr>
<td>InsPoint.h 335</td>
<td>InsPtEnable 335</td>
<td></td>
<td></td>
</tr>
<tr>
<td>InsPtEnabled 336</td>
<td>InsPtGetHeight 336</td>
<td></td>
<td></td>
</tr>
<tr>
<td>InsPtGetPosition 337</td>
<td>InsPtSetHeight 337</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IntlMgr.h 1247, 1255, 1256, 1908</td>
<td>IntlSetRoutineAddress 1256</td>
<td></td>
<td></td>
</tr>
<tr>
<td>intlErrInvalidSelector 1256</td>
<td>IntlGlueGetRoutineAddress 1908</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IntlGetRoutineAddress 1255, 1256</td>
<td>IntlGlueRoutineAddress 1908</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internet library 1839</td>
<td>IntlMgr.h 1247, 1255, 1256, 1908</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IntlSetRoutineAddress 1256</td>
<td>IoFlags SSL Attribute 2197</td>
<td>IoSetSerialMode 1380</td>
<td></td>
</tr>
<tr>
<td>ioGetScanningMode 1379</td>
<td>IoStruct SSL Attribute 2198</td>
<td>ioSupported 1380</td>
<td></td>
</tr>
<tr>
<td>IoTimeout SSL Attribute 2197</td>
<td>IoLib.h 1373</td>
<td>IrGetScanningMode 1378</td>
<td></td>
</tr>
<tr>
<td>IR Library 2310</td>
<td>IrLocalBusy 1393</td>
<td>IrGetStatistics 1378</td>
<td></td>
</tr>
<tr>
<td>IR manager 1373</td>
<td>IrMaxRxSize 1394</td>
<td>IrIasQueryCallback 1409</td>
<td></td>
</tr>
<tr>
<td>IrAdvanceCredit 1383</td>
<td>IrMaxTxSize 1395</td>
<td>IrIsIrLapConnected 1392</td>
<td></td>
</tr>
<tr>
<td>IrBind 1384</td>
<td>IrOpen 1395</td>
<td>IrIsMediaBusy 1392</td>
<td></td>
</tr>
<tr>
<td>IrCallbackParms 1377</td>
<td>irRestoreScanning 1378</td>
<td>IrIsNoProgress 1393</td>
<td></td>
</tr>
<tr>
<td>IrClose 1385</td>
<td>irSetBaudMask 1379</td>
<td>IrIsRemoteBusy 1393</td>
<td></td>
</tr>
<tr>
<td>IrConnectIrLap 1385</td>
<td>IrSetConTypeLMP 1396</td>
<td>irlib.h 1373</td>
<td></td>
</tr>
<tr>
<td>IrConnectReq 1386</td>
<td>IrSetConTypeTTP 1397</td>
<td>IrLocalBusy 1393</td>
<td></td>
</tr>
<tr>
<td>IrConnectRsp 1388</td>
<td>IrSetDeviceInfo 1397</td>
<td>IrMaxRxSize 1394</td>
<td></td>
</tr>
<tr>
<td>IrDataReq 1389</td>
<td>IrTestReq 1398</td>
<td>IrMaxTxSize 1395</td>
<td></td>
</tr>
<tr>
<td>IrDisconnectIrLap 1390</td>
<td>IrUnbind 1399</td>
<td>iterate structure 111</td>
<td></td>
</tr>
<tr>
<td>IrGetStatistics 1378</td>
<td>Japanese feature set 2318</td>
<td>J</td>
<td></td>
</tr>
<tr>
<td>IrIAS_GetInteger 1401</td>
<td>J</td>
<td>K</td>
<td></td>
</tr>
<tr>
<td>IrIAS_GetIntLsap 1401</td>
<td>kAttnCommandCustomEffect 106</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrIAS_GetObjectID 1402</td>
<td>kAttnCommandDrawDetail 106</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrIAS_GetOctetString 1403</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrIAS_GetOctetStringLen 1403</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrIAS_GetType 1404</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrIAS GetUserString 1404</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrIAS GetUserStringCharSet 1404</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrIAS GetUserStringLen 1405</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrIAS Next 1406</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrIAS_Query 1406</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrIAS SetDeviceName 1408</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrIAS StartResult 1409</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrIasQueryCallback 1409</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrIsIrLapConnected 1392</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrIsMediaBusy 1392</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrIsNoProgress 1393</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrIsRemoteBusy 1393</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrOpen 1395</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrRestoreScanning 1378</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrTestReq 1398</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrUnbind 1399</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
kAttnCommandDrawList 106
kAttnCommandGoThere 106
kAttnCommandGoToIt 106
kAttnCommandIterate 107
kAttnCommandPlaySound 106
kAttnCommandSnooze 107
kAttnFlagsAllBits 112
kAttnFlagsAlwaysCustomEffect 112
kAttnFlagsAlwaysLED 112
kAttnFlagsAlwaysSound 112
kAttnFlagsAlwaysVibrate 112
kAttnFlagsCapabilitiesMask 117
kAttnFlagsCustomEffectBit 112
kAttnFlagsEverything 113
kAttnFlagsHasCustomEffect 118
kAttnFlagsHasLED 118
kAttnFlagsHasSound 118
kAttnFlagsHasVibrate 118
kAttnFlagsLEDBit 112
kAttnFlagsNoCustomEffect 113
kAttnFlagsNoLED 113
kAttnFlagsNoSound 113
kAttnFlagsNothing 113
kAttnFlagsNoVibrate 113
kAttnFlagsSoundBit 112
kAttnFlagsUserSettingsMask 117
kAttnFlagsUserWantsCustomEffect 118, 837
kAttnFlagsUserWantsLED 118, 837
kAttnFlagsUserWantsSound 118, 837
kAttnFlagsUserWantsVibrate 118, 837
kAttnFlagsUseUserSettings 112
kAttnFlagsVibrateBit 112
kAttnFtrCapabilities 117
kAttnFtrCreator 117
kAttnLevelInsistent 115
kAttnLevelSubtle 115
kCncDeviceKindLocalNetwork 1272
kCncDeviceKindModem 1272
kCncDeviceKindPhone 1272
kCncDeviceKindSerial 1272
kCncErrDBAccessFailed 1289, 1290, 1291, 1292, 1293, 1294, 1295
kCncErrProfileParamNotFound 1289, 1290, 1291, 1292, 1296
kCncFtrCncMgrVersion 2323
kCncMgrVersion 2323
kCncNotifyCreateRequest 76
kCncNotifyDeleteRequest 76
kCncNotifyModifyRequest 76
kCncNotifyUpdateListRequest 77
kCncParam_PSDCreator 1268, 1271
kCncParam_PSDName 1268, 1271
kCncParam_PSDParameterBuffer 1268
kCncParam_PSDType 1268, 1271
kCncParamBaud 1264, 1270
kCncParamBluetoothDeviceAddr 1264, 1270
kCncParamBluetoothDeviceName 1264, 1270
kCncParamBuffer 1273
kCncParamCountryIndex 1264, 1265, 1270
kCncParamDeviceKind 1264, 1265, 1271
kCncParamDialingMode 1264, 1265, 1271
kCncParamFlowControl 1271
kCncParamInitString 1271
kCncParamIntlModemCountryStringList 1266
kCncParamIntlModemResetStringList 1266, 1269
kCncParamInvisible 1267, 1271
kCncParamLocked 1267, 1271
kCncParamName 1267, 1271
kCncParamNoDetails 1267, 1271
kCncParamNonEditable 1268, 1271
kCncParamPort 1268, 1271
kCncParamReadOnly 1269, 1271
kCncParamReceiveTimeOut 1269, 1271
kCncParamResetString 1269, 1271
kCncParamSerialPortFlags 1269
kCncParamString 1273
kCncParamSystemFlag 1273, 1276
kCncParamSystemFlags 1269, 1271, 1273
kCncParamThirdPartiesRange 1275
kCncParamTimeOut 1269, 1271
kCncParamTTCreator 1270, 1272
kCncParamTTType 1270, 1272
kCncParamUInt16 1273
kCncParamUInt32 1273
kCncParamUInt8 1273
kCncParamVersion 1270, 1272
kCncParamVolume 1270, 1272
kCncProfileClassicResetStringSize 1271
kCncProfileNameSize 1271
kCncProfileUsualInitStringSize 1271
kCncProfileUsualResetStringSize 1271
kCncProfileVersion 1270
kCoordinatesDouble 1163
kCoordinatesNative 1163
kCoordinatesOneAndAHalf 1163
kCoordinatesQuadruple 1163
kCoordinatesStandard 1163
kCoordinatesTriple 1163
kDensityDouble 710, 714
kDensityLow 710, 714
kDrvrVersion 1535
kDrvrVersion3 1535
kDrvrVersion4 1535
key manager functions 759–761
key queue
  size 955
keyBitPageDown 759
keyBitPageUp 759
keyBitPower 759
keyboard display 984
KeyCurrentState 759
keyDownEvent 56, 99, 221, 272, 299, 362, 375, 382, 383, 946, 2314
KeyMgr.h 759
KeyRates 760
KeySetMask 761
kHelperNotifyActionCodeEnumerate 749, 752
kHelperNotifyActionCodeExecute 752
kHelperNotifyActionCodeValidate 752, 754
kHelperServiceClassIDEMail 754, 755, 757
kHelperServiceClassIDFax 754, 757
kHelperServiceClassIDSMS 754, 756, 757
kHelperServiceClassIDVoiceDial 754, 757
kMaxCountryNameLen 768, 770
kMaxCurrencyNameLen 768, 770
kMaxCurrencySymbolLen 768, 770, 771
largeBoldFont 712, 736, 1903, 2308
largeFont 711, 736, 1903
LastAlert SSL Attribute 2198
LastApi SSL Attribute 2199
LastIO SSL Attribute 2200
launch codes
  summary 3, 71, 749
  SysBroadcastActionCode 968
launch flags 36
Launcher.h 417
ledFont 712
LEVENT_DATA_IND 1381
LEVENT_DISCOVERY_CNF 1381
LEVENT_LAP_CON_CNF 1381
LEVENT_LAP_CON_IND 1381
LEVENT_LAP_DISCON_IND 1381
LEVENT_LM_CON_CNF 1381
LEVENT_LM_CON_IND 1381
LEVENT_LM_DISCON_IND 1382
LEVENT_PACKET_HANDLED 1382
LEVENT_STATUS_IND 1382
LEVENT_TEST_CNF 1382
LEVENT_TEST_IND 1382
libEvtHookKeyMask 57
libPalmOSGlue.a 1891
LineInfoPtr 202
LineInfoType 203
lines
  erasing 1182
  inverting 1199
list objects
  and pen tracking 62
  creating category list 136
drawItemsCallback 343, 355
fields 341
functions 343–355
structure 340
List.h 339
ListAttrType 339
ListDrawDataFuncType 355
ListPtr 343
lists
  setting items 974
ListType structure 340
lLanguageName constants 767
lmAnyCountry 773
lmAnyLanguage 194, 773, 1071
lmChoice... constants 770
lmErrBadLocaleIndex 769
lmErrBadLocaleSettingChoice 769
lmErrSettingDataOverflow 769
lmErrUnknownLocale 773
LmGetLocaleSetting 772, 773
LmGlueGetLocaleSetting 771, 1892
LmGlueGetNumLocales 772, 1892
LmGlueLocaleToIndex 774, 1892
LmLocaleType 764, 770, 773, 837
local ID 788, 796
    from chunk handle 782
locale 837
LocaleMgr.h 763
Localize.h 763, 1247
LocGetNumberSeparators 16, 765, 1257
locking chunk 779
locking system 34
longDateStrLength 1061
lookup 31
    example 31
LstDrawList 343
lstEnterEvent 57, 58, 299, 347
LstEraseList 344
lstExitEvent 58
LstGetNumberOfItems 344
LstGetSelection 345
LstGetSelectionText 345
LstGetTopItem 346
LstGetVisibleItems 346
LstGlueSetFont 1910
LstGlueGetFont 1910
LstGlueGetItemsText 1910
LstGlueGetTopItem 1892
LstGlueSetFont 1911
LstGlueSetIncrementalSearch 1911
LstHandleEvent 57, 58, 347
LstMakeItemVisible 348
LstNewList 349
LstPopupList 350
LstScrollList 351
lstSelectEvent 58
LstSetDrawFunction 351
LstSetHeight 352
LstSetListChoices 352
LstSetPosition 353
LstSetSelection 354
LstSetTopItem 355

M

maxFieldLines 224
maxFieldTextLen 237
maxStrIToALen 925
MdmDial 1411
mdmErrBusy 1411
mdmErrCmdError 1411
mdmErrNoDCD 1411
mdmErrNoTone 1411
mdmErrUserCan 1411
MdmHangUp 1412
MeasurementSystemType 770
MemCard Info 775
MemCmp 776
MemDebugMode 777
memErrChunkLocked 781
memErrInvalidParam 668, 669, 780, 783, 793
memErrNotEnoughSpace 155, 668, 669, 670, 780,
    962, 968, 1464, 1600
MemHandleCardNo 777
MemHandleDataStorage 778
MemHandleFree 778
MemHandleHeapID 779
MemHandleLock 779
MemHandleNew 780
MemHandleResize 780
MemHandleSetOwner 781
MemHandleSize 782
MemHandleSetOwner 781
MemHandleToLocalID 782
MemHandleUnlock 783
MemHeapCheck 783, 784
MemHeapCompact 784
MemHeapDynamic 784
memHeapFlagReadOnly 785
MemHeapFlags 785
MenuAddItem 367, 371, 375
MenuBarAttrType 359
MenuBarPtr 364
MenuBarType 365, 372
menuItemCause 61, 383
menuChr 372, 381, 383, 384, 388
menuCloseEvent 59
MenuCmdBarAddButton 59, 95, 361, 372, 373
MenuCmdBarButtonType 360, 362
MenuCmdBarDisplay 376
MenuCmdBarButtonData 376
menuCmdBarMaxLength 377
menuCmdBarOnLeft 372
menuCmdBarOnRight 372
menuCmdBarOpenEvent 59, 95, 222, 299, 373, 376, 377, 383
menuCmdBarResultType 375
MenuCmdBarButtonType 362
MenuCmdBarType 360, 363, 366
menuCommandCause 61, 383
MenuDispose 378
MenuDrawMenu 360, 378
MenuEraseStatus 363, 377, 380
menuErrNoMenu 371
menuErrNotfound 372
menuErrOutOfMemory 373
menuErrSameId 372
menuErrToManyItems 373
menuEvent 54, 60, 299, 362, 375, 382, 383
MenuGetActiveMenu 381
MenuHandleEvent 59, 60, 95, 360, 375, 378, 382
MenuHideItem 384
MenuInit 385
menuItemType 367, 372
menuOpenEvent 60, 372, 383, 384, 388
MenuPullDownPtr 369
MenuPullDownType 367, 369
menus
  FrmSetMenu 319
    functions 371–386
MenuSeparatorChar 368, 370, 371
MenuSetActiveMenu 385
MenuSetActiveMenuRscID 386
MenuShowItem 387
missing character symbol 725
modal window 350, 1170, 1201
Mode SSL Attribute 2200
modem 1411
ModemMgr.h 1411
modified field objects 208
multi-byte characters 998, 1001, 1023, 1024, 1025, 1030, 1032
  attributes 1013
  comparison 999, 1014
  converting 1036
  encodings support 997–1040
  searching 1022, 1914, 1918
  size 1012
multiple preferences 847

N
narrowTextTableItem 427, 433, 434, 451
net library
  closing 1424, 1438
  open count 1466
  opening 1463, 1464
netCfgNameCTPWireless 1420
netCfgNameCTPWireline 1420
netCfgNameDefault 1420
netCfgNameDefWireless 1420
netCfgNameDefWireline 1420
netConfigIndexCurSettings 1466
NetConfigNamePtr 1413
NetConfigNameType 1413
netErrAlreadyOpen 1463, 1465
netErrAlreadyOpenWithAnotherConfig 1465
netErrAuthFailure 1459
netErrBadScript 1458
netErrBufTooSmall 1433, 1436, 1450, 1465, 1479
netErrBufWrongSize 1450, 1457, 1479, 1483
netErrConfigAliasErr 1433, 1465
netErrConfigBadName 1434, 1435
netErrConfigCantDelete 1433, 1434, 1465
netErrConfigCantPointToAlias 1427
netErrConfigEmpty 1433, 1465
netErrConfigNotAlias 1426, 1427
netErrConfigNotFound 1430, 1433, 1465
netErrConfigTooMany 1435
netErrDNSSAborted 1440, 1442, 1444
netErrDNSAllocationFailure 1440, 1441, 1443
netErrDNSBadName 1440, 1441, 1443
netErrDNSBadProtocol 1440, 1442, 1444
netErrDNSFormat 1440, 1441, 1443
netErrDNSImpossible 1440, 1442, 1444
netErrDNSIrrelevant 1440, 1442, 1444
netErrDNSLabelTooLong 1440, 1441, 1443
netErrDNSNameTooLong 1440, 1441, 1443
netErrDNSNIY 1440, 1442, 1443
netErrDNSNonexistantName 1440, 1442, 1443
netErrDNSNoPort 1440, 1442, 1444
netErrDNSNoRecursion 1440, 1442, 1444
netErrDNSNoRRS 1440, 1442, 1444
netErrDNSNotInLocalCache 1440, 1442, 1444
netErrDNSRefused 1440, 1442, 1443
netErrDNSServerFailure 1440, 1441, 1443
netErrDSTimeout 1440, 1441, 1443
netErrDNSTruncated 1440, 1442, 1444
netErrDNSUnreachable 1440, 1441, 1443
netErrInterfaceDown 1475, 1477
netErrInterfaceNotFound 1446, 1447, 1448, 1450, 1457, 1458, 1465, 1475, 1477
netErrInternal 1491, 1492
netErrInvalidInterface 1449
netErrInvalidSettingSize 1483, 1496, 1498
netErrIPCanFragment 1475, 1477
netErrIPktOverflow 1475, 1477
netErrIPNoDst 1475, 1477
netErrIPNoRoute 1475, 1477
netErrIPNoSrc 1475, 1477
netErrMessageTooBig 1475, 1477
netErrNoInterfaces 1436, 1463, 1465, 1490, 1492
netErrNoMoreSockets 1494
netErrNoMultiPacketAddr 1502
netErrNoMultiPktAddr 1475, 1477
netErrNotOpen 1425, 1438, 1440, 1441, 1443, 1445, 1448, 1458, 1460, 1468, 1469, 1472, 1474, 1477, 1485, 1486, 1488, 1489, 1490, 1492, 1493, 1496, 1497, 1501, 1502, 1503
netErrOutOfRangeBlocks 1426, 1427, 1429, 1430, 1431, 1433, 1434, 1435, 1436, 1465, 1475, 1477, 1488, 1489, 1491, 1492, 1494, 1502
netErrOutOfMemory 1438, 1463, 1465, 1494
netErrOutOfPackets  1475, 1477
netErrOutOfResources  1492
netErrParamErr  1426, 1427, 1429, 1433, 1434, 1438, 1460, 1465, 1468, 1469, 1474, 1477, 1485, 1486, 1488, 1489, 1490, 1492, 1494, 1496, 1497, 1502
netErrPortInUse  1490, 1492
netErrPPPAddressRefused  1459
netErrPPPTimeout  1458
netErrPrefNotFound  1433, 1449, 1450, 1457, 1463, 1465, 1479
netErrQuietTimeNotElapsed  1490, 1492
netErrReadOnlySetting  1457, 1483
netErrSocketAlreadyConnected  1488, 1491, 1492
netErrSocketBusy  1490, 1492
netErrSocketClosedByRemote  1475, 1477, 1485, 1487, 1488, 1491, 1492
netErrSocketInputShutdown  1502
netErrSocketNotConnected  1475, 1477, 1485
netErrSocketNotListening  1485
netErrSocketNotOpen  1438, 1468, 1469, 1474, 1477, 1485, 1487, 1488, 1489, 1490, 1492, 1496, 1497, 1502
netErrStillOpen  1425
netErrTimeout  1438, 1439, 1440, 1441, 1443, 1445, 1465, 1468, 1469, 1474, 1477, 1485, 1486, 1488, 1489, 1490, 1492, 1493, 1496, 1497, 1501
netErrTooManyInterfaces  1446
netErrTooManyTCPConnections  1491
netErrUnimplemented  1450, 1457, 1460, 1485, 1496, 1497
netErrUnknownProtocol  1445
netErrUnknownService  1445
netErrUnknownSetting  1450, 1457, 1479, 1483
netErrUnreachableDest  1475, 1477
netErrUserCancel  1438, 1458, 1468
netErrWouldBlock  1438, 1468, 1470, 1475, 1477, 1491
netErrWrongSocketType  1485, 1491, 1492, 1496, 1498
netFDIsSet  1472
netFDSet  1472
netFDSetSize  1473
NetFDSetType  1471
netFDZero  1472
NetHostInfoBufType  1414
NetHostInfoType  1414
NetHToNL  1422
NetHToNS  1423
netIFMediaDown  96
netIFMediaUp  96
NetIFSettingEnum  1450, 1451, 1457
netIOFlagDontRoute  1421
netIOFlagOutOfBand  1421
netIOFlagPeek  1421
NetIOParamType  1470
NetIOVecPtr  1470
NetIOVecType  1470
NetIPAddr  1414, 1423, 1424
NetLibAddrAToIN  1423
NetLibAddrINToA  1424
NetLibClose  1424
NetLibConfigAliasGet  1426
NetLibConfigAliasSet  1427
NetLibConfigDelete  1429
NetLibConfigIndexFromName  1430
NetLibConfigList  1431
NetLibConfigMakeActive  1432
NetLibConfigRename  1434
NetLibConfigSaveAs  1435
NetLibConnectionRefresh  1436
NetLibDmReceive  1437
NetLibFinishCloseWait  1438
NetLibGetHostByAddr  1439
NetLibGetHostByName  1441
NetLibGetMailExchangeByName  1442
NetLibGetServByName  1444
NetLibIFAAttach  1446
NetLibIFDetach  1447
NetLibIFDown  1448
NetLibIFGet  1449
NetLibIFSettingGet  1450
NetLibIFSettingSet  1457
NetLibIFUp  1458
NetLibMaster  1459
NetLibOpen  1463
NetLibOpenConfig  1464
NetLibOpenCount  1466
NetLibReceive 1467, 1508
NetLibReceivePB 1469
NetLibSelect 1471
NetLibSend 1474, 1509
NetLibSendPB 1476
NetLibSettingGet 1479
NetLibSettingSet 1483
NetLibSocketAccept 1484, 1485
NetLibSocketAddr 1486
NetLibSocketAddrEnum 1487
NetLibSocketClose 1489
NetLibSocketConnect 1490, 1508
NetLibSocketListen 1491, 1492
NetLibSocketOpen 1493, 1508
NetLibSocketOptionGet 1495
NetLibSocketOptionSet 1497
NetLibSocketShutdown 1501
NetLibTracePrintF 1502
NetLibTracePutS 1503
NetMasterEnum 1459
netMasterICMPStats command 1462
netMasterInterfaceInfo command 1460
netMasterInterfaceStats command 1461
netMasterIPStats command 1462
NetMasterPBPtr 1459
netMasterTCPSstats command 1462
netMasterTraceEventGet command 1462
netMasterUDPStats command 1462
NetMgr.h 1413
NetNToHL 1504
NetNToHS 1504
NetServInfoBufType 1416
NetServInfoType 1416
NetSettingEnum 1479, 1483
NetSocket.c 1507
NetSocketAddrEnum 1417
NetSocketAddrINET 1494
NetSocketAddrINType 1417
netSocketAddrRaw 1494
NetSocketAddrRawType 1418
NetSocketAddrType 1419
netSocketDirBoth 1501
NetSocketDirEnum 1501
netSocketDirInput 1501
netSocketDirOutput 1501
NetSocketLingerType 1499
NetSocketOptEnum 1495, 1497, 1498
NetSocketOptLevelEnum 1495, 1497, 1498
netSocketProtoIPRAW 1493
netSocketProtoIPTCP 1493
netSocketProtoIPUDP 1493
NetSocketRef 1493
NetSocketTypeEnum 1419
netTracingAppMsgs 1421
netTracingAppPktIP 1421
netTracingData 1422
netTracingData40 1422
netTracingErrors 1421
netTracingFuncs 1421
netTracingIFHi 1422
netTracingIFLow 1422
netTracingIFMid 1422
netTracingMsgs 1421
netTracingPktts 1421
NetUReadN 1507
NetUTCPOpen 1508
NetUWriteN 1509
new serial manager 2320, 2321
NFNT resource 718
nfnt resource 721
nilEvent 62, 959, 960
noFocus 272, 274, 287, 317
noListSelection 345
noMenuItemSelection 366, 370
noMenuSelection 366, 370
noPreferenceFound 841, 843
noteTextTableItem 434
notification manager 2330
notifyDetailsP 94
NotifyMgr.h 71, 801
NumberFormatType 764, 770
numericGraffitiSilkscreenArea 954
numericTableItem 425
numLockMask 57
numUneditableCategories 136, 139, 147
O
off-screen windows 1168
olume Constants 884
omErrBadOverlayDBName 819
omErrBaseRequiresOverlay 569
omErrDatabaseRequiresOverlay 614
omErrInvalidLocaleIndex 814
omErrNoNextSystemLocale 815
omErrUnknownLocale 569, 818, 819
omFtrCreator 812
omFtrDefaultLocale 812
omFtrShowErrorsFlag 812
OmGetCurrentLocale 813
OmGetIndexedLocale 814
OmGetNextSystemLocale 815
OmGetRoutineAddress 816
OmGetSystemLocale 773, 817
OmGlueGetCurrentLocale 813, 1892
OmGlueGetSystemLocale 817, 819
OmLocaleToOverlayDBName 818
OmLocaleType 764, 811
OmOverlayDBNameToLocale 819
omOverlayDBType 613
omOverlayRscID 812
omOverlayRscType 812
OmSelector 816
OmSetSystemLocale 820
open count of net library 1466
opening net library 1463, 1464
optionKeyMask 57
OverlayMgr.h 811, 816
overlays 613

P
Palm OS 2.0 feature set 2304
Palm OS 3.0 feature set 2308
Palm OS 3.1 feature set 2312
Palm OS 3.2 feature set 2315
Palm OS 3.5 feature set 2324, 2330, 2342, 2346, 2351
PalmLocale.h 763, 765, 767
PalmOSGlue.lib 1891, 2313, 2316
panel list (SysCreatePanelList) 972
password functions 823
Password.h 823
PatternType 1153
PceNativeCall 1258
PdiDefineReaderDictionary 2253
PdiDefineResizing 2254
PdiDefineWriterDictionary 2255
PdiEnterObject 2256
PdiLibClose 2257
PdiLibOpen 2257
PdiParameterPairTest 2258
PdiReaderDelete 2259
PdiReaderNew 2259
PdiReaderType 2238
PdiReadParameter 2260
PdiReadProperty 2261
PdiReadPropertyField 2262
PdiReadPropertyName 2264
PdiSetCharset 2265
PdiSetEncoding 2266
PdiWriteBeginObject 2266
PdiWriteEndObject 2267
PdiWriteParameter 2268
PdiWriteParameterStr 2270
PdiWriteProperty 2271
PdiWritePropertyBinaryValue 2272
PdiWritePropertyFields 2273
PdiWritePropertyStr 2274
PdiWritePropertyValue 2275
PdiWriterDelete 2276
PdiWriterNew 2276
PdiWriterType 2240
PeerCert SSL Attribute 2202
PeerCommonName SSL Attribute 2202
pen
   current status 950
pen manager functions 825–826
pen queue
   flushing 949
   size 956
PenBtnInfoType 951
PenCalibrate 825
penDownEvent 44, 47, 49, 53, 57, 62, 66, 171, 221, 299, 347, 382, 412
PenMgr.h 825

Palm OS Programmer’s API Reference 2387
penMoveEvent 62
PenResetCalibration 826
penUpEvent 63, 945
PhoneLookup.h 493
PhoneNumberLookup 498
PhoneNumberLookupCustom 499
PilotMain 995
PixelFormatType 533
PluginCallbackProcType 1511
PluginCmdPtr 1512
PluginCmdType 1512
PluginExecCmdType 1513, 1515
PluginInfoPtr 1514
PluginInfoType 1514, 1515
pluginMaxNumOfCmds 1514
pluginNetLibCallUPrc 1516, 1519, 1521
pluginNetLibCheckCancelStatus 1516, 1520
pluginNetLibConnLog 1516, 1521
pluginNetLibDoNothing 1516, 1519
pluginNetLibGetSerLibRefNum 1517, 1521
pluginNetLibGetUserName 1516, 1520
pluginNetLibGetUserPwd 1516, 1520
pluginNetLibPromptUser 1516, 1520
pluginNetLib.ReadByte 1516, 1519
pluginNetLibWriteBytes 1516, 1520
PointType 853
popSelectEvent 63, 298, 299
popup list 350
popupTriggerTableItem 425
Port... constants 1537
poweredOnKeyMask 57
prefAlarmSoundVolume 875
preferenceDataVer2 841
preferenceDataVer3 841
preferenceDataVer4 841
preferenceDataVer5 841
preferenceDataVer6 841
preferenceDataVer8 841
preferenceDataVer9 841
preferenceDataVerLatest 841, 2314
preferenceDataVersion 2314
preferences
  changing with launch codes 32
  multiple application preferences 847
Preferences.h 763, 764, 827
prefGameSoundVolume 875
PrefGetAppPreferences 842, 2306
PrefGetAppPreferencesV10 844
PrefGetPreference 94, 845
PrefGetPreferences 845
PrefOpenPreferenceDB 846
PrefOpenPreferenceDBV10 847
PrefSetAppPreferences 847
PrefSetAppPreferencesV10 849
PrefSetPreference 850
PrefSetPreferences 850
prefShowPrivateRecords 390, 391
prefSysSoundVolume 875
prefTimeZone 194
PrgCallbackData 400
PrgCallbackFunc 400
PrgHandleEvent 393
PrgStartDialog 394
PrgStartDialogV31 396
PrgStopDialog 397
PrgUpdateDialog 398
PrgUserCancel 399
printable characters 1009
printf 903
PrivateRecords.h 389
privateRecordViewEnum 389, 835
procedure alarms 508
progress manager 2309
Progress Manager callback function 400
Progress.h 393
ProtocolVersion SSL Attribute 2203
putc 904
putchar 904
puts 905
PwdExists 823
PwdRemove 823
PwdSet 824
PwdVerify 824
query callback function 1409
R

RAM-based heaps 791
RbufSize SSL Attribute 2204
RctCopyRectangle 854
RctGetIntersection 854
RctInsetRectangle 855
RctOffsetRectangle 856
RctPtlnRectangle 857
RctSetRectangle 857
read callback function 1353
ReadBufPending SSL Attribute 2204
ReadOutstanding SSL Attribute 2205
ReadProc 1315
ReadRecPending SSL Attribute 2205
ReadStreaming SSL Attribute 2205
records
    deleting 584
    detaching 585
    ID 587
    retrieving information 622
Rect.h 853
RectanglePtr 855
rectangles
    copying 854
    erasing 1183
    intersecting 854
    moving 856
    resizing 855
    scrolling 1229
RectangleType 853, 855
reinitializing dynamic memory heap 989
repeat control object
    and ctlRepeatEvent 44
repeating button 44
reset 35, 989
ResGlueLoadConstant 501, 1892
ResLoadConstant 501
ResLoadForm 502
ResLoadMenu 502
resource database (SysCurAppDatabase) 973
resource ID 561
resource type 562, 589
resources
    retrieving 600
retrieving information 628
searching for 629
resumeSleepChr 99, 100
RGBColorType 534
ROM-based heaps 785, 791
ROM-based records 618, 620
S

Sampled Sound Application-Defined
Functions 896
Sampled Sound Functions 884
Sampled Sound Structures, Constants, and Data Types 879
SclDrawScrollBar 410
sclEnterEvent 64, 300, 412
sclExitEvent 65, 412
SclGetScrollBar 410
SclHandleEvent 64, 65, 411
sclRepeatEvent 65, 300, 412
    and sclExitEvent 65
SclSetScrollBar 412
scptLaunchCmdDoNothing 1515
scptLaunchCmdExecuteCmd 3, 1513, 1515
scptLaunchCmdListCmds 3, 1514, 1515
ScrDisplayMode 1228, 2311, 2327
ScrDisplayModeOperation 1228
ScreenAttrType 1222
ScriptPlugin.h 1511
ScriptPluginLaunchCodesEnum 1515
ScriptPluginSelectorProcPtr 1518
scroll arrows
    FrmUpdateScrollers 323
scroll position in field 214
scrollbar functions 410–415
scrollbar objects
    fields 407
    in tables 448
    structure 407
ScrollBar.h 405
ScrollBarAttrType 405
ScrollBarPtr 406
ScrollBarRegionType 406
ScrollBarType 407
scrolling rectangle in window 1229
ScrOperation 1162
searching for string 252
secret records, removing 626
SecSelectViewStatus 390
Security application 34
SecurityAutoLockType 827
SecVerifyPW 391
SelDay.h 185
SelectDay 188, 2306
selectDayByDay 188
selectDayByMonth 188
selectDayByWeek 188
SelectDayV10 189
selection in field 215
SelectOneTime 189
SelectTime 191
SelectTimeZone 193
SelTime.h 185
SelTimeZone.h 185
separatorItemSelection 370
SerClearErr 1595, 1598
SerClose 1596
SerCtlEnum 1593
serCtlBreakStatus (in SerCtlEnum) 1594
serCtlEmuSetBlockingHook (in SerCtlEnum) 1594
serCtlFirstReserved (in SerCtlEnum) 1593
serCtlHandshakeThreshold (in SerCtlEnum) 1594
serCtlLAST (in SerCtlEnum) 1594
serCtlMaxBaud (in SerCtlEnum) 1594
serCtlStartBreak (in SerCtlEnum) 1593
serCtlStartLocalLoopback (in SerCtlEnum) 1594
serCtlStopBreak (in SerCtlEnum) 1593
serCtlStopLocalLoopback (in SerCtlEnum) 1594
serDev... constants 1559
serErrAlreadyOpen 1596, 1600
serErrBadParam 1597, 1600, 1609
serErrBadPort 1585
serErrConfigurationFailed 1578, 1585
serErrLineErr 1595, 1596, 1599, 1601, 1602, 1603, 1604
serErrNotOpen 1585, 1586, 1587, 1596, 1597, 1598, 1609
serErrNotSupported 1585, 1586, 1587
serErrStillOpen 1596
serErrTimeOut 1585, 1601, 1602, 1604, 1605, 1606, 1607
serFncConsole 1529, 1556
serFncDebugger 1529, 1556
serFncHotSync 1529, 1556
serFncPPPSession 1529, 1556
serFncSLIPSession 1529, 1556
serFncTelephony 1529, 1556
serFncUndefined 1529, 1556
SerGetSettings 1598
SerGetStatus 1598
serial capabilities constants 1559
Serial Library 1600
serial port feature constants 1537
serial settings constants 1560
serial status constants 1562
SerialDrvr.h 1523
SerialLinkMgr.h 1611
SerialMgr.h 1551
SerialMgrOld.h 1593
SerialVdrv.h 1523
serLineError... constants 1599
SerOpen 1599
serPortConsolePort 1557
serPortCradlePort 1557
serPortCradleRS232Port 1558
serPortCradleUSBPort 1558
serPortIrPort 1557
serPortLocalHotSync 1557
SerReceive 1601
SerReceive10 1602
SerReceiveCheck 1603
SerReceiveFlush 1603, 1604
SerReceiveWait 1604
SerSend 1605
SerSend10 1606
SerSendFlush 1607
SerSendWait 1607
SerSetReceiveBuffer 1608
SerSetSettings 1608
SerSettingsType 1594
servent 1416
SessionReused SSL Attribute 2206
sethostname 1481
shiftKeyMask 57
silk-screen buttons
  EvtGetPenBtnList 951
SilkscreenAreaType 953
silkscreenRectGraffiti 954
silkscreenRectScreen 954
SioAddCommand 905
SioClearScreen 915
SioExecCommand 915
Siofgetc 899, 902, 906
Siofgets 900, 906
Siofprintf 900, 907
Siofputc 901, 904, 908
Siofputs 902, 908
SioFree 916
Siogets 903, 909
SioHandleEvent 916
SioInit 917
SioMain 918
Sioprintf 903, 909
Sioputs 905, 910
Siosystem 910, 912
Siovfprintf 911, 913
SleepEventParamType 100
SliderControlType 164
SlkClose 1611
SlkCloseSocket 1612
slkErrAlreadyOpen 1613
slkErrBadParam 1617
slkErrBuffer 1615
slkErrChecksum 1615
slkErrNotOpen 1611
slkErrOutOfSockets 1614
slkErrSocketNotOpen 1612, 1613, 1615, 1616, 1617, 1618, 1619
slkErrTimeOut 1615, 1616
slkErrWrongDestSocket 1615
SlkFlushSocket 1612
SlkOpen 1613
SlkOpenSocket 1614
SlkPktHeaderType 1618
SlkReceivePacket 1615
SlkSendPacket 1616
SlkSetSocketListener 1617
SlkSocketListenerType 1617, 1618
SlkSocketPortID 1618
SlkSocketSetTimeout 1619
SlkWriteDataType 1617
slLib 2164
SmsLib.h 2219
SmsParamsType 2219
SmsPrefType 2222
SmsReceiveCDMAParamsType 2223
SmsReceiveGSMParamsType 2225
SmsReceiveParamsType 2225
SmsReceiveTDMAParamsType 2227
SMSReportParamsType 2227
SmsSendCDMAParamsType 2228
SMSSendParamsType 2230
SmsSendTDMAParamsType 2232
sndAlarm 876
sndAlarmVolume 884
SndBlockingFuncType 877
SndCallbackInfoType 860
sndClick 876
sndCmdFreqDurationAmp 870
sndCmdFreqOn 870
SndCmdIDType 861
sndCmdNoteOn 871
sndCmdQuiet 871
SndCommandType 861
SndComplFuncType 878
sndConfirmation 876
SndCreateMidiList 868
SndDoCmd 869
sndError 876
sndFlagAsync 883
sndFlagNormal 883
sndFlagSync 883
sndFloat 880
sndFloatBig 880
sndFloatLittle 880
sndFloatOpposite 880
sndGameVolume 884
SndGetDefaultVolume 871
sndInfo 876  
sndInput 882  
sndInt16 880  
sndInt16Big 880  
sndInt16Little 880  
sndInt16Opposite 880  
sndInt32 880  
sndInt32Big 880  
sndInt32Little 880  
sndInt32Opposite 880  
sndInt8 880  
SndInterruptSmfIrregardless 872  
sndMaxAmp 833, 862  
SndMidiListItemType 862  
sndMidiNameLength 863  
SndMidiRecHdrType 863  
sndMidiRecSignature 864  
sndMono 883  
sndOutput 882  
sndPanCenter 883  
sndPanFullLeft 883  
sndPanFullRight 883  
SndPlayResource 884  
SndPlaySMF 872  
SndPlaySmfIrregardless 874  
SndPlaySmfResource 875  
SndPlaySmfResourceIrregardless 876  
SndPlaySystemSound 876  
SndPtr 879  
SndSampleType 879  
SndSampleTypeTag 880  
SndSetDefaultVolume 877  
SndSmfCallbacksType 864  
SndSmfChanRangeType 865  
SndSmfCmdEnum 865  
SndSmfOptionsType 865  
SndSmfPlayAllMilliSec 867  
sndStartUp 876  
sndStereo 883  
SndStreamBufferCallback 896  
SndStreamDelete 889  
SndStreamGetPan 890  
SndStreamGetVolume 890  

SndStreamMode 881  
SndStreamModeTag 881  
SndStreamPause 891  
SndStreamRef 882  
SndStreamSetVolume 893  
SndStreamStart 894  
SndStreamStop 895  
SndStreamWidth 882  
SndStreamWidthTag 882  
SndSysBeepType 867  
sndSystemVolume 884  
sndUInt8 880  
sndWarning 876  
sockaddr 1419  
sockaddr_in 1417  
socket 1508  
socket listener 1616  
socket listener procedure 1616, 1617, 1618  
Socket SSL Attribute 2207  
soft reset 35, 989  
sorting array elements 983  
sorting text 559  
SortRecordInfoType 562  
Sound Manager 859  
sound manager 2309  
Sound Resource Playback Flags 883  
SoundMgr.h 859  
sprintf 912  
sprintf (StrPrintF) 934  
SrmClearErr 1563  
SrmClose 1563  
SrmControl 1552, 1558, 1560, 1564, 1569  
srmControlCustom 1567  
SrmCtlEnum 1552, 1564  
SrmCustomControl 1567  
srmDefaultCTSTimeout 1569  
SrmExtOpen 1557, 1568, 1570, 1575  
SrmExtOpenBackground 1557, 1570, 1576  
SrmGetDeviceCount 1571  
SrmGetDeviceInfo 1551, 1570  
SrmGetDeviceInfo 1551, 1572  
SrmGetStatus 1562, 1573, 1578, 1579, 1581  
SrmOpen 1557, 1570, 1574  
SrmOpenBackground 1557, 1575
Palm OS Programmer's API Reference

SrmOpenConfigType  1555, 1568, 1570
SrmPrimeWakeupHandler  1576, 1590
SrmRcvQType  1527
SrmReceive  1577, 1582, 1584, 1590
SrmReceiveCheck  1579, 1582
SrmReceiveFlush  1579
SrmReceiveWait  1581, 1584
SrmReceiveWindowClose  1582, 1591
SrmReceiveWindowOpen  1583, 1591
SrmSend  1584, 1587
SrmSendCheck  1585
SrmSendFlush  1586
SrmSendWait  1587
SrmSetReceiveBuffer  1569, 1581, 1588
srmSettings... constants  1560
SrmSetWakeupHandler  1577, 1589, 1590
srmStatus... constants  1562
SSL Alerts  2214
SSL Attribute Data Types  2182
SSL Attributes  2187
SSL Attributes and Macros  2181
SSL Certificate Errors  2216
SSL Cryptography Errors  2215
SSL Data Types  2163
SSL Function Protocol Errors  2213
SSL Handshake Errors  2215
SSL Illegal Message Errors  2216
SSL Macro Names  2181
SSL Structures  2166
SSL Structures and Data Types  2163
SslAttribute  2163
SslCallback  2166
SslCipherSuiteInfo  2167
SslContext  2163
SslContextGet_AppInt32  2187
SslContextGet_AppPtr  2188
SslContextGet_AutoFlush  2188
SslContextGet_BufferedReuse  2189
SslContextGet_CipherSuite  2190
SslContextGet_CipherSuiteInfo  2190
SslContextGet_CipherSuites  2191
SslContextGet_Compat  2192
SslContextGet_DontSendShutdown  2193
SslContextGet_DontWaitForShutdown  2193
SslContextGet_Error  2194
SslContextGet_HsState  2194
SslContextGet_InfoCallback  2195
SslContextGet_IOFlags  2197
SslContextGet_IoStruct  2198
SslContextGet_IoTimeout  2197
SslContextGet_LastAlert  2198
SslContextGet_LastApi  2199
SslContextGet_LastIO  2200
SslContextGet_Mode  2201
SslContextGet_PeerCert  2202
SslContextGet_ProtocolVersion  2203
SslContextGet_RbufSize  2204
SslContextGet_ReadBufPending  2204
SslContextGet_ReadOutstanding  2205
SslContextGet_ReadRecPending  2205
SslContextGet_ReadStreaming  2206
SslContextGet_SessionReused  2206
SslContextGet_Socket  2207
SslContextGet_SslSession  2208
SslContextGet_SslVerify  2208
SslContextGet_SslVerifyCallback  2209
SslContextGet_WriteBufPending  2210
SslContextSet_AppInt32  2187
SslContextSet_AppPtr  2188
SslContextSet_AutoFlush  2188
SslContextSet_CipherSuites  2191
SslContextSet_Compat  2192
SslContextSet_DontSendShutdown  2193
SslContextSet_DontWaitForShutdown  2193
SslContextSet_Error  2194
SslContextSet_InfoCallback  2195
SslContextSet_IOFlags  2197
SslContextSet_IoStruct  2198
SslContextSet_IoTimeout  2197
SslContextSet_Mode  2201
SslContextSet_ProtocolVersion  2203
SslContextSet_RbufSize  2204
SslContextSet_ReadStreaming  2206
SslContextSet_Socket  2207
SslContextSet_VerifyCallback 2209
SslContextSet_WbufSize 2210
sslErrBadArgument 2213
sslErrBadDecode 2216
sslErrBadLength 2214
sslErrBadOption 2214
sslErrBadPeerFinished 2215
sslErrBadSignature 2216
sslErrBufferTooSmall 2214
sslErrCbAbort 2214
sslErrCert 2216
sslErrCertDecodeError 2216
sslErrCsp 2215
sslErrDivByZero 2215
sslErrEof 2214
sslErrExtraHandshakeData 2215
sslErrFailed 2214
sslErrFatalAlert 2214
sslErrHandshakeEncoding 2215
sslErrHandshakeProtocol 2215
sslErrIo 2214
sslErrNoModInverse 2215
sslErrNoRandom 2215
sslErrNullArg 2214
sslErrOutOfMemory 2214
sslErrReadAppData 2215
sslErrRecordError 2216
sslErrUnexpectedRecord 2216
sslErrUnsupportedCertType 2216
sslErrUnsupportedSignatureType 2216
sslErrVerifyBadSignature 2216
sslErrVerifyConstraintViolation 2216
sslErrVerifyNotAfter 2216
sslErrVerifyNotBefore 2217
sslErrVerifyNoTrustedRoot 2217
sslErrVerifyUnknownCriticalExtension 2217
sslErrWrongMessage 2216
SslExtendedItem 2169
SslExtendedItems 2171
SslIoBuf 2172
SslLibCallback 2166, 2173, 2174
SslLibGet_AppInt32 2187
SslLibGet_AppPtr 2188
SslLibGet_AutoFlush 2188
SslLibGet_CIPHERSuites 2191
SslLibGet_Compat 2192
SslLibGet_DontSendShutdown 2193
SslLibGet_DontWaitForShutdown 2193
SslLibGet_InfoCallback 2195
SslLibGet_Mode 2201
SslLibGet_ProtocolVersion 2203
SslLibGet_RbufSize 2204
SslLibGet_ReadStreaming 2206
SslLibGet_VerifyCallback 2209
SslLibGet_WbufSize 2210
SslLibSet_AppInt32 2187
SslLibSet_AppPtr 2188
SslLibSet_AutoFlush 2188
SslLibSet_CIPHERSuites 2191
SslLibSet_Compat 2192
SslLibSet_DontSendShutdown 2193
SslLibSet_DontWaitForShutdown 2193
SslLibSet_InfoCallback 2195
SslLibSet_Mode 2201
SslLibSet_ProtocolVersion 2203
SslLibSet_RbufSize 2204
SslLibSet_ReadStreaming 2206
SslLibSet_VerifyCallback 2209
SslLibSet_WbufSize 2210
SsslSession 2174
SsslSession_SSL_Attribute 2207
SsslSocket 2176
SsslVerify 2177
SsslVerify_SSL_Attribute 2208
standard IO functions 899
StartApplication
   and PrefGetPreferences 846
stat 690
stderr 711, 736, 1903
StdIOPalm.h 899
StdIOProvider.h 899
Stereo Pan Constants 883
StrATol 919
StrCaselessCompare 920
StrCat 921
strcat function substitute (StrCat) 921
StrChr 921
strchr function substitute (StrChr) 921
strcmp function substitute (StrCompare) 922
StrCompare 922
StrCompareAscii 923
StrCopy 924
strcpy function substitute (StrCopy) 924
StrDelocalizeNumber 924
StrDelocalizeNumber, and launch code 16
Streaming SSL Attribute 2209
stricmp function substitute (StrCaselessCompare) 920
string manager 919–940
string resource
   copying 969
string searching 252
StringMgr.h 919
StrToA 925
StrToH 926
StrLen 926
strlen function substitute (StrLen) 926
StrLocalizeNumber 927
   launch code 16
StrNCaselessCompare 929
StrNCat 930
strncat function substitute (StrNCat) 930
strncpy 932
StrNCmp 931
StrNCmpAscii 932
StrNCopy 933
strokes, translating 956
StrPrintF 912, 934
StrStr 935
strstr function substitute (StrStr) 935
StrToLower 935
structure of field object 198
StrVPrintF 914, 936
summary of launch codes 3, 71, 749
symbol11Font 712
symbol7Font 712
symbolFont 711
sys_socket.h 1507
SysAlarmTriggeredParamType 9, 510
SysAppLaunch 962
sysAppLaunchCmdAddRecord 6
sysAppLaunchCmdAlarmTriggered 9, 17, 507
sysAppLaunchCmdCardLaunch 15
SysAppLaunchCmdCardType 16
sysAppLaunchCmdCountryChange 16
sysAppLaunchCmdDisplayAlarm 9, 16, 507
sysAppLaunchCmdExgAskUser 17, 19, 1297,
   1322, 1335, 1337, 2308
sysAppLaunchCmdExgGetData 19
sysAppLaunchCmdExgPreview 19, 22, 1310, 1323,
   1336, 1342
sysAppLaunchCmdExgReceiveData 21, 22, 1310,
   1317, 1320, 1323, 1342, 2308
sysAppLaunchCmdFind 22, 1918
sysAppLaunchCmdGoto 23, 26, 51, 1298, 1320,
   1334, 2308
sysAppLaunchCmdGoToURL 27, 1349
sysAppLaunchCmdHandleSyncCallApp 28
sysAppLaunchCmdHandleSyncCallAppType 29
sysAppLaunchCmdInitDatabase 30
sysAppLaunchCmdLookup 31, 2304
sysAppLaunchCmdNotify 31, 74, 807
sysAppLaunchCmdOpenDB 32
sysAppLaunchCmdPanelCalledFromApp 32, 33,
   2304
sysAppLaunchCmdReturnFromPanel 32, 33, 2304
sysAppLaunchCmdSaveData 23, 33
sysAppLaunchCmdSyncNotify 34, 1344
sysAppLaunchCmdSystemLock 34, 2304
sysAppLaunchCmdSystemReset 35, 97
sysAppLaunchCmdTimeChange 35
sysAppLaunchCmdURLParams 36
SysAppLauncherDialog 417
sysAppLaunchFlagNewGlobals 26
sysAppLaunchFlagNewGlobals launch flag 37
sysAppLaunchFlagSubCal launch flag 37
sysAppLaunchFlagUIApp launch flag 37
SysBatteryInfo 963, 2311
SysBatteryInfoV20 964
SysBatteryKind 831
SysBinarySearch 966
SysBroadcastActionCode 968
SysCopyStringResource 969
SysCreateDataBaseList 970
SysCreatePanelList 972
SysCurAppDatabase 973
SysDBListItemType 970
SysDisplayAlarmParamType 17
sysErrLibNotFound 985, 986
sysErrNoFreeLibSlots 986
sysErrNoFreeRAM 986
sysErrOutOfOwnerID 962
sysErrOutOfOwnerIDs 968
sysErrParamErr 962, 968, 987
SysErrString 973
SysEvent.h 39, 2328
SysEvtMgr.h 941
sysExternalConnectorAttachEvent 77
sysExternalConnectorDetachEvent 77
SysFatalAlert 418
sysFileCBIConnectPanelHelper 1559
sysFileCSmsLib 2220
sysFileCUart328 1558
sysFileCUart328EZ 1558
sysFileCVirtIrComm 1559
sysFileCVirtRfComm 1559
sysFileDescStdIn 1472
SysFormPointerArrayToStrings 974, 1329, 1330
sysFtrNewSerialPresent 2321
sysFtrNewSerialVersion 2322
sysFtrNumEncoding 2318
sysFtrNumIntlMgr 2316
sysFtrNumNotifyMgrVersion 802, 2330
sysFtrNumProcessor328 2314
sysFtrNumProcessorEZ 2314
sysFtrNumProcessorID 2314
sysFtrNumProcessorIS68K 975
sysFtrNumProcessorISARM 976
sysFtrNumProcessorMask 2314
sysFtrNumROMVersion 2304, 2308, 2312
SysGetOSVersionString 977, 2311
SysGetRomToken 977
SysGetToken 2311
SysGetStackInfo 978, 2311
SysGetTrapAddress 979
SysGlueGetTrapAddress 979, 1892
SysGlueTrapExists 1912
SysGraffitiReferenceDialog 418
SysGremlins 980, 2311
SysHandleEvent 78, 89, 93, 98, 99, 808, 981
SysInsertionSort 981
SysKeyboardDialog 984, 2306
SysKeyboardDialogV10 984
SysLibFind 985
SysLibInstall 985
SysLibLoad 986
SysLibRemove 987
sysNotifyAntennaRaisedEvent 78
SysNotifyBroadcast 802
SysNotifyBroadcastDeferred 362, 804
sysNotifyBroadcasterCode 75, 801
SysNotifyBroadcastFromInterrupt 805
sysNotifyCardInsertedEvent 79
sysNotifyCardRemovedEvent 80
sysNotifyDBChangedEvent 82
SysNotifyDBChangedType 83
SysNotifyDBCreatedEvent 81
SysNotifyDBCreatedType 81
sysNotifyDBDeletedEvent 85, 583
SysNotifyDBDeletedType 85
sysNotifyDBDirtyEvent 86
SysNotifyDBDirtyType 87
SysNotifyDBInfoType 88
sysNotifyDefaultQueueSize 801, 803
sysNotifyDeleteProtectedEvent 87
sysNotifyDeviceUnlocked 88, 93
SysNotifyDisplayChangeDetailsType 89
sysNotifyDisplayChangeEvent 88, 1213
sysNotifyEarlyWakeupEvent 89
sysNotifyErrBroadcastBusy 802
sysNotifyErrDuplicateEntry 807
sysNotifyErrEntryNotFound 809
sysNotifyErrNoStackSpace 802
sysNotifyErrQueueFull 804, 805
sysNotifyForgotPasswordEvent 90
sysNotifyGotUsersAttention 91
sysNotifyHelperEvent 91, 749, 751
sysNotifyIRDSniffEvent 73
sysNotifyLateWakeupEvent 88, 93
sysNotifyLocaleChangedEvent 94
SysNotifyLocaleChangedType 94
sysNotifyMenuCmdBarOpenEvent 59, 95, 299, 373, 377
sysNotifyNetLibIFMediaEvent 95
SysNotifyNetLibIFMediaType 96
sysNotifyNoDatabaseID 801
sysNotifyNormalPriority 802, 807
SysNotifyParamType 32, 74, 75, 807
sysNotifyPhoneEvent 73
sysNotifyPOSEMountEvent 73
SysNotifyProcPtr 806, 808, 810
SysNotifyRegister 806
sysNotifyResetFinishedEvent 97
sysNotifyRetryEnqueueKey 97
sysNotifySleepNotifyEvent 98, 99
sysNotifySleepRequestEvent 98, 99
sysNotifySyncFinishEvent 34, 101
sysNotifySyncStartEvent 34, 101
sysNotifyTimeChangeEvent 35, 101, 1069
SysNotifyUnregister 809
sysNotifyVersionNum 802
sysNotifyVolumeMountedEvent 102
sysNotifyVolumeUnmountedEvent 103
SysQSort 987
SysRandom 989
sysRandomMax 989
SysReset 989
sysResIDExtPrefs 614
sysResTExtPrefs 614
SysSetAutoOffTime 990
SysSetTrapAddress 990
sysSleepAutoOff 100
sysSleepPowerButton 100
sysSleepResumed 100
sysSleepUnknown 100
SysStringByIndex 991
SysTaskDelay 992
system 912
system events
    checking availability 959
system keyboard display 984
SystemMgr.h 3, 961
SystemPreferencesChoice 829, 837
SystemPreferencesType 829
SystemResources.h 1551
SysTicksPerSecond 992
sysTrap.... 979, 990
SysTraps.h 979, 990
SysUIAppSwitch 993
SysUtils.h 961

T

table functions 433–475
table objects
    fields 430
    structure 430
Table.h 419
TableAttrType 419
 TableColumnAttrType 420
TableDrawItemFuncPtr 433, 451
TableDrawItemFuncType 475
TableItemPtr 422
TableItemStyleType 424
TableItemType 422
TableLoadDataFuncType 433, 451, 476
tableMaxTextItemSize 424
TablePtr 427
TableRowAttrType 428
tables
    setting load data callback 467
    setting save data callback 473
TableSaveDataFuncType 478
TableType 430
tAIB 535
taif 535
TblDrawTable 433
TblEditing 434
tblEnterEvent 66, 67, 300
TblEraseTable 435
tblExitEvent 67, 68
TblFindRowData 435
TblFindRowID 436
TblGetBounds 436
TblGetColumnSpacing 437
TblGetColumnWidth 437
TblGetCurrentField 438, 446
TblGetItemBounds 438
TblGetItemFont 439, 2311
TblGetItemInt 440
TblGetItemPtr 440
TblGetLastUsableRow 441
TblGetNumberOfRows 442
TblGetRowData 443
TblGetRowHeight 443
TblGetRowID 444
TblGetSelection 444
TblGlueGetColumnMasked 1912
TblGlueGetNumberOfColumns 442, 1893
TblGlueGetTopRow 445, 1893
TblGlueSetSelection 474, 1893
TblGrabFocus 445
TblHandleEvent 66, 67, 447
TblHasScrollBar 448
TblInsertRow 449
TblMarkRowInvalid 450
TblMarkTableInvalid 450
TblRedrawTable 451
TblReleaseFocus 452
TblRemoveRow 453
TblRowInvalid 453
TblRowMasked 454
TblRowSelectable 455
TblRowUsable 455
tblSelectEvent 67, 448
TblSelectItem 456
TblSetBounds 457
TblSetColumnEditIndicator 457
TblSetColumnMasked 458
TblSetColumnSpacing 459
TblSetColumnUsable 460
TblSetColumnWidth 460
TblSetCustomDrawProcedure 461
TblSetItemFont 462, 2311
TblSetItemInt 463
TblSetItemPtr 464
TblSetItemStyle 465
TblSetLoadDataProcedure 467
TblSetRowData 467
TblSetRowHeight 468
TblSetRowID 469
TblSetRowMasked 470
TblSetRowSelectable 471
TblSetRowStaticHeight 472
TblSetRowUsable 472
TblSetSaveDataProcedure 473
TblUnhighlightSelection 475
tblUnusableRow 441
tbmf 535
Tbmp 535
TelCallStateType 1625
TelCancel 1638
TelCfgGetPhoneNumber 1685
TelCfgGetPhoneNumberType 1681
TelCfgGetSmsCenterType 1682
TelCfgSetSmsCenter 1688
TelCfgGetSmsCenter 1687
TelClose 1639
TelClosePhoneConnection 1640
TelDataCallNumberType 1715
TelDtcCallNumber 1720
TelDtcCloseLine 1722
TelDtcReceiveData 1723
TelDtcReceiveDataType 1716
TelDtcSendData 1724
TelDtcSendDataType 1717
TelEmcCall 1726
TelEmcCloseLine 1728
TelEmcGetNumber 1729
TelEmcGetNumberType 1717
TelEmcSelectNumber 1732
TelEmcSetNumber 1734
TelEmcSetNumberType 1718
TelephoneMgr.h 2224, 2228, 2229
TelEventType 1623
TelGetCallState 1641
TelGetEvent 1643
TelGetNumberCount 1731
TelGetTelephonyEvent 1644
TelInfGetInformation 1645
TelInformationType 1627
TelIsCfgServiceAvailable 1647
TelIsDtcServiceAvailable 1648
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TelIsEmcServiceAvailable</td>
<td>1649</td>
</tr>
<tr>
<td>TelIsInfServiceAvailable</td>
<td>1650</td>
</tr>
<tr>
<td>TelIsNwkServiceAvailable</td>
<td>1652</td>
</tr>
<tr>
<td>TelIsOemServiceAvailable</td>
<td>1653</td>
</tr>
<tr>
<td>TelIsPhbServiceAvailable</td>
<td>1654</td>
</tr>
<tr>
<td>TelPhoneConnected</td>
<td>1655</td>
</tr>
<tr>
<td>TelIsPowServiceAvailable</td>
<td>1656</td>
</tr>
<tr>
<td>TelIsSmsServiceAvailable</td>
<td>1658</td>
</tr>
<tr>
<td>TelIsSndServiceAvailable</td>
<td>1659</td>
</tr>
<tr>
<td>TelIsSpcServiceAvailable</td>
<td>1660</td>
</tr>
<tr>
<td>TelIsStyServiceAvailable</td>
<td>1661</td>
</tr>
<tr>
<td>TelMatchPhoneDriver</td>
<td>1663</td>
</tr>
<tr>
<td>TelNwkGetLocation</td>
<td>1699</td>
</tr>
<tr>
<td>TelNwkGetLocationType</td>
<td>1695</td>
</tr>
<tr>
<td>TelNwkGetNetworkName</td>
<td>1701</td>
</tr>
<tr>
<td>TelNwkGetNetworkNameType</td>
<td>1696</td>
</tr>
<tr>
<td>TelNwkGetNetworks</td>
<td>1703</td>
</tr>
<tr>
<td>TelNwkGetNetworksType</td>
<td>1697</td>
</tr>
<tr>
<td>TelNwkGetNetworkType</td>
<td>1705</td>
</tr>
<tr>
<td>TelNwkGetSearchMode</td>
<td>1706</td>
</tr>
<tr>
<td>TelNwkGetSelectedNetwork</td>
<td>1269, 1707</td>
</tr>
<tr>
<td>TelNwkGetSignalLevel</td>
<td>1709</td>
</tr>
<tr>
<td>TelNwkSelectNetwork</td>
<td>1711</td>
</tr>
<tr>
<td>TelNwkSetSearchMode</td>
<td>1712</td>
</tr>
<tr>
<td>TelOemCall</td>
<td>1664</td>
</tr>
<tr>
<td>TelOemCallType</td>
<td>1628</td>
</tr>
<tr>
<td>TelOpen</td>
<td>1665</td>
</tr>
<tr>
<td>TelOpenPhoneConnection</td>
<td>1666</td>
</tr>
<tr>
<td>TelPhbAddEntry</td>
<td>1821</td>
</tr>
<tr>
<td>TelPhbDeleteEntry</td>
<td>1823</td>
</tr>
<tr>
<td>TelPhbEntryType</td>
<td>1815</td>
</tr>
<tr>
<td>TelPhbGetAvailablePhonebooks</td>
<td>1824</td>
</tr>
<tr>
<td>TelPhbGetAvailablePhonebooksType</td>
<td>1817</td>
</tr>
<tr>
<td>TelPhbGetEntries</td>
<td>1826</td>
</tr>
<tr>
<td>TelPhbGetEntriesType</td>
<td>1818</td>
</tr>
<tr>
<td>TelPhbGetEntry</td>
<td>1828</td>
</tr>
<tr>
<td>TelPhbGetEntryCount</td>
<td>1830</td>
</tr>
<tr>
<td>TelPhbGetEntryCountType</td>
<td>1819</td>
</tr>
<tr>
<td>TelPhbGetEntryMaxSizes</td>
<td>1832</td>
</tr>
<tr>
<td>TelPhbGetEntryMaxSizesType</td>
<td>1819</td>
</tr>
<tr>
<td>TelPhbGetSelectedPhonebook</td>
<td>1833</td>
</tr>
<tr>
<td>TelPhbSelectPhonebook</td>
<td>1835</td>
</tr>
<tr>
<td>TelPowGetBatteryStatus</td>
<td>1667</td>
</tr>
<tr>
<td>TelPowGetPowerLevel</td>
<td>1669</td>
</tr>
<tr>
<td>TelPowSetPhonePower</td>
<td>1670</td>
</tr>
<tr>
<td>TelSendCommandString</td>
<td>1671</td>
</tr>
<tr>
<td>TelSendCommandStringType</td>
<td>1628</td>
</tr>
<tr>
<td>TelSmsDateTimeType</td>
<td>1755</td>
</tr>
<tr>
<td>TelSmsDeleteMessage</td>
<td>1785</td>
</tr>
<tr>
<td>TelSmsDeleteMessageType</td>
<td>1756</td>
</tr>
<tr>
<td>TelSmsDeliveryAdvancedCDMAType</td>
<td>1757</td>
</tr>
<tr>
<td>TelSmsDeliveryAdvancedGSMType</td>
<td>1758</td>
</tr>
<tr>
<td>TelSmsDeliveryAdvancedTDMAType</td>
<td>1759</td>
</tr>
<tr>
<td>TelSmsDeliveryMessageType</td>
<td>1761</td>
</tr>
<tr>
<td>TelSmsExtensionType</td>
<td>1764, 1768, 1769, 1780</td>
</tr>
<tr>
<td>TelSmsGetAvailableStorage</td>
<td>1786</td>
</tr>
<tr>
<td>TelSmsGetAvailableStorageType</td>
<td>1765</td>
</tr>
<tr>
<td>TelSmsGetDataMaxSize</td>
<td>1788</td>
</tr>
<tr>
<td>TelSmsGetMessageCount</td>
<td>1790</td>
</tr>
<tr>
<td>TelSmsGetMessageCountType</td>
<td>1766</td>
</tr>
<tr>
<td>TelSmsGetSelectedStorage</td>
<td>1792</td>
</tr>
<tr>
<td>TelSmsGetUniquePartId</td>
<td>1793</td>
</tr>
<tr>
<td>TelSmsManualAckType</td>
<td>1767</td>
</tr>
<tr>
<td>TelSmsReadMessage</td>
<td>1795</td>
</tr>
<tr>
<td>TelSmsReadMessages</td>
<td>1798</td>
</tr>
<tr>
<td>TelSmsReadMessagesType</td>
<td>1769</td>
</tr>
<tr>
<td>TelSmsReadReport</td>
<td>1800</td>
</tr>
<tr>
<td>TelSmsReadReports</td>
<td>1802</td>
</tr>
<tr>
<td>TelSmsReadReportsType</td>
<td>1770</td>
</tr>
<tr>
<td>TelSmsReadSubmittedMessage</td>
<td>1804</td>
</tr>
<tr>
<td>TelSmsReadSubmittedMessages</td>
<td>1806</td>
</tr>
<tr>
<td>TelSmsReadSubmittedMessagesType</td>
<td>1771</td>
</tr>
<tr>
<td>TelSmsReportType</td>
<td>1772</td>
</tr>
<tr>
<td>TelSmsSelectStorage</td>
<td>1808</td>
</tr>
<tr>
<td>TelSmsSendManualAcknowledge</td>
<td>1809</td>
</tr>
<tr>
<td>TelSmsSendMessage</td>
<td>1811</td>
</tr>
<tr>
<td>TelSmsSendMessageType</td>
<td>1774</td>
</tr>
<tr>
<td>TelSmsSubmitAdvancedCDMAType</td>
<td>1774</td>
</tr>
<tr>
<td>TelSmsSubmitAdvancedGSMType</td>
<td>1775</td>
</tr>
<tr>
<td>TelSmsSubmitAdvancedTDMAType</td>
<td>1777</td>
</tr>
<tr>
<td>TelSmsSubmitMessageType</td>
<td>1778</td>
</tr>
<tr>
<td>TelSmsSubmittedMessageType</td>
<td>1780</td>
</tr>
<tr>
<td>TelSndMute</td>
<td>1672</td>
</tr>
<tr>
<td>TelSndPlayKeyTone</td>
<td>1673</td>
</tr>
<tr>
<td>TelSndPlayKeyToneType</td>
<td>1629</td>
</tr>
<tr>
<td>TelSndStopKeyTone</td>
<td>1675</td>
</tr>
</tbody>
</table>
TelSpcAcceptCall 1735
TelSpcCallNumber 1737
TelSpcCloseLine 1739
TelSpcConference 1741
TelSpcGetCallerNumber 1742
TelSpcGetCallerNumberType 1718
TelSpcHoldLine 1744
TelSpcPlayDTMF 1745
TelSpcPlayDTMFType 1719
TelSpcRejectCall 1747
TelSpcRetrieveHeldLine 1748
TelSpcSelectLine 1749
TelSpcSendBurstDTMF 1750
TelSpcStartContinuousDTMF 1752
TelSpcStopContinuousDTMF 1753
TelStyChangeAuthenticationCode 1690
TelStyChangeAuthenticationType 1683
TelStyEnterAuthenticationCode 1691
TelStyGetAuthenticationState 1693
text clipboard 206
text manager 997–1040, 2317
text, finding with GetCharCaselessValue 558
TextMgr.h 997
textTableItem 426, 433, 434, 451
textWithNoteTableItem 426, 433, 434, 451
TimAdjust 1066
TimDateTimeToSeconds 1067
time manager
  structures 1045
time system resource 185
time, displaying and selecting 191
TimePtr 1054
timeTableItem 427
TimeToAscii 1070
TimeType 1054
timeZoneStringLength 1071
TimGetSeconds 1067
TimGetTicks 1068
TimSecondsToDate 1068
TimSetSeconds 101, 1069
TimTimezoneToUTC 1072
TimUTCToTimeZone 1073
tint 501
titles
  active area 53
  copying form title 277
  transliteration 1036
  translitOpLowerCase 1037
  translitOpPreprocess 1037
  translitOpType 1037
  translitOpUpperCase 1037
  TsmGlueGetFepMode 1892
  TsmGlueSetFepMode 1892
  TxtByteAttr 998
  TxtCaselessCompare 999
  and StrCaselessCompare 920, 922, 929, 931
TxtCharAttr 1000
TxtCharBounds 1001
TxtCharEncoding 1003
TxtCharIsAlpha 1004
TxtCharIsAlpha 1004
TxtCharIsAlpha 1004
TxtCharIsAlpha 1004
TxtCharIsCntrl 1005
TxtCharIsDelim 1005
TxtCharIsDigit 1006
TxtCharIsGraph 1006
TxtCharIsHex 1008
TxtCharIsLower 1011
TxtCharIsPrint 1010
TxtCharIsSpace 1010
TxtCharIsUpper 1011
TxtCharIsValid 1011
TxtCharSize 1012
TxtCharWidth 1013
TxtCharXAttr 1013
TxtCompare 1014
TxtConvertEncoding 1017
TxtEncodingName 1021
txtErrConvertOverflow 1018
txtErrTransitOverflow 1037
txtErrTransitOverrun 1037
txtErrTransitUnderflow 1037
txtErrUnknownEncoding 1019
txtErrUnknownTranslitOp 1037
TxtFindString 24, 1022, 1914
  and FindString 252
TxtGetChar 1023
TxtGetNextChar 1024, 1030
TxtGetPreviousChar 1025
TxtGetTruncationOffset 1027
TxtGetWordWrapOffset 1027
TxtGlueByteAttr 998, 1893
TxtGlueCaselessCompare 1000, 1893
TxtGlueCharAttr 1001, 1893
TxtGlueCharBounds 1002, 1893
TxtGlueCharEncoding 1004, 1893
TxtGlueCharIsAlpha 1005, 1893
TxtGlueCharIsCtrl 1005, 1893
TxtGlueCharIsDelim 1006, 1893
TxtGlueCharIsDigit 1006, 1893
TxtGlueCharIsGraph 1007, 1893
TxtGlueCharIsHex 1008, 1893
TxtGlueCharIsLower 1008, 1893
TxtGlueCharIsPrint 1009, 1893
TxtGlueCharIsPunct 1010, 1893
TxtGlueCharIsSpace 1010, 1893
TxtGlueCharIsUpper 1011, 1893
TxtGlueCharIsValid 1012, 1893
TxtGlueCharIsVirtual 1913
TxtGlueCharSize 1012, 1893
TxtGlueCharWidth 1893
TxtGlueCharXAttr 1014, 1894
TxtGlueCompare 1016, 1894
TxtGlueEncodingName 1021, 1894
TxtGlueFindString 24, 252
TxtGlueGetChar 1024, 1894
TxtGlueGetHorizEllipsisChar 1915, 2313
TxtGlueGetNextChar 1025, 1894
TxtGlueGetNumericSpaceChar 1916, 2313
TxtGlueGetPreviousChar 1026, 1894
TxtGlueGetTruncationOffset 1027, 1894
TxtGlueLowerChar 1916
TxtGlueLowerChar 1916
TxtGlueMaxEncoding 1029, 1894
TxtGlueNextCharSize 1031, 1894
TxtGlueParamString 1032, 1894
TxtGluePrepFindString 1022, 1914, 1918
TxtGluePreviousCharSize 1033, 1894
TxtGlueReplaceStr 1034, 1894
TxtGlueSetNextChar 1035, 1894
TxtGlueStrEncoding 1036, 1894
TxtGlueStripSpaces 1919
TxtGlueTransliterate 1038, 1894
TxtGlueTruncateString 1920
TxtGlueUpperChar 1921
TxtGlueUpperCase 1922
TxtGlueWordBounds 1039, 1894
TxtMaxEncoding 1028
TxtNameToEncoding 1029
TxtNextCharSize 1030
TxtParamString 1031
TxtPreviousCharSize 1032
TxtReplaceStr 1033
TxtSetNextChar 1034
TxtStrEncoding 1035
TxtTransliterate 1036
TxtWordBounds 1039

U

UDABufferSize 2279
UDAControl 2284
UDADelete 2286
UDAEEndOfReader 2286
UDAEExchangeReaderNew 2291
UDAEExchangeWriterNew 2291
UDAFilterJoin 2287
UDAFilterType 2280
UDAInitiateWrite 2287
UDAMemoryReaderNew 2284, 2292
UDAMoreData 2288
UDAOBJECTTYPE 2280
UDARead 2288
UDAREADERTYPE 2281
UDAWRITERFLUSH 2289
UDAWRITERJOIN 2290
UDAWRITERTYPE 2282
UIBrightnessAdjust 489
UIColor.h 479
UIColorGetTableEntryIndex 483
UIColorGetTableEntryRGB 485
UIColorSetTableEntry 487
UIColorTableEntries 479
UICommon.h 493
UIContrastAdjust 490
UIControls.h 489
UIPickColor 490
UIPickColorStartPalette 491
UIPickColorStartRGB 491
UIPickColorStartType 490
UIResources.h 374
UIResources.r 614
UnderlineModeType 197, 1154
unitsEnglish 827
unitsMetric 827
user name
   obtaining 1251

V
valid characters 1011
vchrCommand 372, 381, 383, 384, 388
vchrHardAntenna 2320
vchrMenu 299, 372, 381, 383, 384, 388, 2328
vchrRadioCoverageFail 2320
vchrRadioCoverageOK 2320
VdrvAPIType structure 1527
VDrvClose 1539
VDrvConfigType 1528
VDrvControl 1540
VDrvCtlOpCodeEnum 1530
VDrvCustomControl 1542
VDrvOpen 1543
VDrvStatus 1545
VDrvWrite 1545
VerifyCallback SSL Attribute 2209
vfprintf 913
VFS Manager 1075
VFSAnyMountParamType 103, 1076
VFSCustomControl 1085
VFS DirCreate 1087
VFS DirEntryEnumerate 1088
vfsErrBadData 1083
vfsErrBadName 1083
vfsErrBufferOverflow 1083
vfsErrDirectoryNotFound 1083
vfsErrDirNotEmpty 1083
vfsErrFileAlreadyExists 1084
vfsErrFileBadRef 1084
vfsErrFileEOF 1084
vfsErrFileGeneric 1084
vfsErrFileNotFoundException 1084
vfsErrFilePermissionDenied 1084
vfsErrFileStillOpen 1084
vfsErrIsADirectory 1084
vfsErrNameShortened 1084
vfsErrNoFileSystem 1084
vfsErrNotADirectory 1084
vfsErrVolumeBadRef 1084
vfsErrVolumeFull 1085
vfsErrVolumeStillMounted 1085
VFSExportDatabaseToFile 1090
VFSExportDatabaseToFileCustom 1091
VFSExportProcPtr 1145
vfsFileAttrArchive 1082
vfsFileAttrDirectory 1082
vfsFileAttrHidden 1082
vfsFileAttrLink 1082
vfsFileAttrReadOnly 1082
vfsFileAttrSystem 1082
vfsFileAttrVolumeLabel 1082
VFSFileClose 1093
VFSFileCreate 1094
VFSFileDBGetRecord 1095
VFSFileDBGetResource 1097
VFSFileDBInfo 1099
VFSFileDelete 1102
VFSFileEOF 1103
VFSFileGetAttributes 1104
VFSFileGetDate 1105
VFSFileOpen 1106
VFSFileRead 1108
VFSFileReadData 1109
VFSFileRename 1111
VFSFileResize 1113
VFSFileSeek 1114
VFSFileSetAttributes 1115
VFSFileSetDate 1117
VFSFileSize 1118
WinDrawGrayRectangleFrame 1176
WinDrawInvertedChars 1176
WinDrawLine 1177
WinDrawOperation 1160
WinDrawPixel 1178
WinDrawRectangle 1178
WinDrawRectangleFrame 1179
WinDrawTruncChars 1180
winEnterEvent 68, 69, 313
WinEraseChars 1181
WinEraseLine 1182
WinErasePixel 1182
WinEraseRectangle 1183
WinEraseRectangleFrame 1184
WinEraseWindow 1184
winExitEvent 69
WinFillLine 1185
WinFillRectangle 1185
WinGetActiveWindow 1186
WinGetBitmap 1186
WinGetClip 1188
WinGetCoordinateSystem 1188
WinGetDisplayExtent 1189
WinGetDisplayWindow 1189
WinGetDrawWindow 1190
WinGetFirstWindow 1191
WinGetFramesRectangle 1191
WinGetPattern 1192
WinGetPatternType 1193
WinGetPixel 1193
WinGetPixelRGB 1194
WinGetSupportedDensity 1195
WinGetWindowBounds 1187
WinGetWindowExtent 1196
WinGetWindowFrameRect 1197
WinGlueDrawChar 1174, 1894
WinGlueDrawTruncChars 1181, 1894
WinGlueGetFrameType 1923
WinGlueSetFrameType 1923
WinHandle 1162
WinIndexToRGB 1197
WinInvertChars 1198
WinInvertLine 1199
WinInvertPixel 1199
WinInvertRectangle 1200
WinInvertRectangleFrame 1201
WinLineType 1162
WinLockInitType 1223
WinModal 1201
WinPaintBitmap 1202
WinPaintChar 1203
WinPaintChars 1204
WinPaintLine 1205
WinPaintLines 1205
WinPaintPixel 1206
WinPaintPixels 1207
WinPaintRectangle 1207
WinPaintRectangleFrame 1208
WinPaintRoundedRectangleFrame 1209
WinPaintTiledBitmap 1210
WinPalette 89, 1211
WinPopDrawState 1213
WinPtr 1163
WinPushDrawState 1214
WinResetClip 1214
WinRestoreBits 1215
WinRGBToIndex 1215
WinSaveBits 1217
WinScaleCoord 1218
WinScalePoint 1219
WinScaleRectangle 1220
WinScreenGetAttribute 1221
WinScreenLock 1223
WinScreenMode 89, 1224
WinScreenModeOperation 1224
WinScreenUnlock 1229
WinScrollRectangle 1229
WinSetActiveWindow 68, 1230
WinSetBackColor 1231
WinSetBackColorRGB 1232
WinSetBounds 1233
WinSetClip 1233
WinSetCoordinateSystem 1234
WinSetDrawMode 1234
WinSetDrawWindow 1235
WinSetForeColor 1236
WinSetForeColorRGB 1237
WinSetPattern 1238
WinSetPatternType 1238
WinSetTextColor 1239
WinSetTextColorRGB 1240
WinSetUnderlineMode 1241
WinUnscaleCoord 1241
WinUnscalePoint 1242
WinUnscaleRectangle 1243
WinUseTableIndexes 1212
WinValidateHandle 1244
WinWindowToDisplayPt 1245
wireless internet feature set 2318
WirelessIndicator.h 1888
word wrap 734
write callback function 1354
WriteBlock 1548
WriteBufPending SSL Attribute 2210
WriteByte 1548
WriteProc 1318, 1354