PALM OS

Welcome to

Developing Palm OS Applications

Part I: System and User Interface Management

Navigate this online document as follows:

To see bookmarks	Type Command-7
To see information on Adobe Acrobat Reader	Type Command-?
To navigate	Click on any blue hypertext link any Table of Contents entry arrows in the menu bar

PALM OS

U.S. Robotics®

Developing Palm OS™ Applications

Part I

Canada
Metrowerks Inc.
1500 du College, suite 300
St. Laurent, QC
H4L 5G6 Canada

voice: (514) 747-5999
fax: (514) 747-2822

U.S.A. and International
Metrowerks Corporation
2201 Donley Drive
Suite 310
Austin, TX 78758

voice: (512) 873-4700
fax: (512) 873-4900

U.S. Robotics, Palm Computing Division
Mail Order
1-800-881-7256
Metrowerks Mail Order
voice: (800) 377-5416
fax: (512) 873-4901

U.S. Robotics, Palm Computing Division World Wide Web site:	`http://www.usr.com/palm`
Metrowerks World Wide Web site (Internet):	`http://www.metrowerks.com`
Registration information (Internet):	`register@metrowerks.com`
Technical support (Internet):	`support@metrowerks.com`
Sales, marketing, & licensing (Internet):	`sales@metrowerks.com`
AppleLink:	`METROWERKS`
America OnLine:	`goto: METROWERKS`
Compuserve:	`goto: METROWERKS`

Table of Contents

Table of Contents v

1 Developing Palm OS Applications 27

Overview of Application Development 27
  Designing UI and Program Functionality 28
    Designing Screen Layout and User Interaction 28
  Constructing UI Resources 29
  Using Managers and Filling out the Program Logic 29
    Using Events and Launch Codes 29
    Using the Palm OS Managers 30
  Building, Debugging, and Testing 30
  Building the Application and Running it on the Device . . . 31
  Using Other Components of the SDK 31
Internal Structure of an Application 31
Naming Conventions 33
Basic Hardware 34
  RAM and ROM 34
  Palm OS Modes of Operation 34
  Palm OS Connectivity. 35
  Real-Time Clock and Timer 35
  Palm OS Device Screen and Sound Generation 36
  Palm OS Device Reset Switch 36

2 Application Control Flow 37

How Events Control an Application. 37
  Basic Application Stages. 38
  The Startup Routine 38
  The Event Loop 41
  The Stop Routine. 44
How Action Codes Control the Application 47
  Action Code Example. 47
  Responding to Action Codes. 49
  Predefined Action Codes 50
    Action Code Flags 51

Action Code Parameter Blocks 51
sysAppLaunchCmdSaveData 51
sysAppLaunchCmdSystemReset. 52
sysAppLaunchCmdInitDatabase. 52
sysAppLaunchCmdSyncCallApplication 52
sysAppLaunchCmdGoto Command 53
sysAppLaunchCmdFind 54
Creating Your Own Action Codes 55

3 Palm OS User Interface Resources 57
Menu Bar Resource 59
Menu Resource. 59
Application Icon Name Resource 62
Alert Resource 63
Button Resource 65
Check Box Resource. 67
Field Resource 70
Form Bitmap Resource 72
Form Resource 73
Gadget Resource 77
Graffiti Shift Resource 78
Label Resource 78
List Resource. 79
Popup List Resource 81
Popup Trigger Resource 81
Push Button Resource 83
Repeating Button Resource. 86
Selector Trigger Resource 88
String Resource. 90
Table Resource 90
Title Resource 91
Version Number String 92
Supported Fonts 92

4 Palm OS Events 93
appStopEvent 93

ctlEnterEvent 93
ctlExitEvent 93
ctlRepeatEvent 94
ctlSelectEvent 94
daySelectEvent 95
fldChangedEvent 95
fldEnterEvent 96
fldHeightChangedEvent 96
frmCloseEvent 96
frmLoadEvent 97
frmOpenEvent 97
frmSaveEvent 97
frmUpdateEvent 98
keyDownEvent 99
lstEnterEvent. 100
lstExitEvent 100
lstSelectEvent 101
menuEvent 101
nilEvent 102
penDownEvent. 102
penMoveEvent 102
penUpEvent 103
popSelectEvent. 103
tblEnterEvent 104
tblExitEvent 104
tblSelectEvent 105
winEnterEvent 106
winExitEvent 106

5 Palm OS UI Objects 107

Control Objects. 107
  Control Object Overview 108
  Control Object Events. 109
  Structure of a Control 110
    Fields of a ControlType Structure 111
  Associated Resources 112

Control Functions 112
Field Objects 113
Field Object Overview 113
Field Object Events 114
Structure of a Field 115
Fields of a Field Structure 116
Associated Resources 117
Field Functions 118
Form Objects 119
Form Object Overview 119
Structure of a Form 121
Fields of Form Objects 124
Associated Resource 125
Form Functions 125
List Object 127
List Object Overview 128
List Object Events 128
Structure of a List 129
List Object Fields 129
Associated Resources 130
List Functions 131
Menu Objects 131
Menu Object Overview 131
Menu Events 132
Structure of a Menu 133
Menu Object Fields. 134
Menu Pull-Down Fields 135
Menu Item Fields 135
Associated Resources 135
Menu Functions 136
Date and Time UI Objects 136
Date and Time Functions 136
Insertion Point Object 136
Insertion Point Functions 136
Table Objects 137
Table Events 137

Structure of a Table 137
Fields of a Table Structure 139
Associated Resource 139
Table Functions 140
Window Objects 141
Window Events 141
Structure of a Window 142
Fields of a Window Structure 142
Window Functions 144

6 Using Palm OS Managers 147
The System Manager 148
System Boot and Reset 149
Power Management 150
Palm OS Power Modes 150
Guidelines for Application Developers 151
The Microkernel 151
Application Support 152
Launching and Clean-Up 152
Event Processing. 152
Inter-Application Communication 155
Using the System Manager 155
System Reset Calls 155
Power Management Calls 156
Application Utilities 156
System Manager Functions 157
The Feature Manager 158
Feature Manager Overview 158
Using the Feature Manager 159
Feature Manager Functions 160
The String Manager 160
The Time Manager 161
Using Real-Time Clock Functions. 161
Using System Ticks Functions 162
Time Manager Structures 162
Time Manager Function Summary 163

The System Event Manager 164
Event Translation: Pen Strokes to Key Events. 164
Pen Queue Management 165
Key Queue Management 166
Auto-Off Control 166
System Event Manager Function Summary 167
The Pen Manager 167
Pen Manager Functions 168
The Key Manager 168
Key Manager Functions 168
The Graffiti Manager 169
Graffiti Manager Function Summary 170
The Alarm Manager. 171
Alarm Manager Overview 171
Using the Alarm Manager 172
Alarm Manager Function Summary 172
The Alert Manager 173
The Alert Resource 173
Alert Manager Functions 173
The Sound Manager. 174
Using the Sound Manager 174
Sound Manager Function Summary 174
The Error Manager 175
Displaying Development Errors 175
The Try and Catch Mechanism 176
Using the Error Manager 177
Using the Try and Catch Mechanism 178
Error Manager Function Summary 179

7 Control, Field, and Insertion Point Functions 181
Control Functions. 181
CtlDrawControl 181
CtlEraseControl 181
CtlGetLabel 182
CtlGetValue 182
CtlHandleEvent 183

CtlHideControl 184
CtlHitControl 184
CtlEnabled 185
CtlSetEnabled 185
CtlSetLabel 186
CtlSetUsable. 186
CtlSetValue 187
CtlShowControl 187
Field UI Functions 188
FldCalcFieldHeight. 188
FldCompactText 188
FldCopy 189
FldCut 189
FldDelete 190
FldDirty. 190
FldDrawField 191
FldEraseField 191
FldFreeMemory 192
FldGetAttributes 192
FldGetBounds 193
FldGetFont 193
FldGetInsPtPosition 194
FldGetMaxChars 194
FldGetScrollPosition 195
FldGetSelection 195
FldGetTextAllocatedSize 196
FldGetTextHandle 196
FldGetTextHeight 197
FldGetTextLength 197
FldGetTextPtr 197
FldGetVisibleLines 198
FldGrabFocus 198
FldHandleEvent 199
FldInsert 200
FldMakeFullyVisible 200
FldPaste. 201

FldRecalculateField 201
FldReleaseFocus 202
FldScrollable. 202
FldScrollField 203
FldSendChangeNotification 203
FldSendHeightChangeNotification 204
FldSetAttributes 204
FldSetBounds 205
FldSetDirty 205
FldSetFont. 206
FldSetInsPtPosition 206
FldSetMaxChars 207
FldSetScrollPosition 207
FldSetSelection. 208
FldSetText 209
FldSetTextAllocatedSize 210
FldSetTextHandle 210
FldSetTextPtr 211
FldSetUsable 211
FldUndo 212
FldWordWrap 212
Insertion Point Functions 213
InsPtEnable 213
InsPtEnabled 213
InsPtGetHeight 214
InsPtGetLocation 214
InsPtSetHeight 215
InsPtSetLocation 215
Functions for System Use Only 216
InsPtCheckBlink 216
InsPtInitialize 216

8 Form, List, and Menu Functions 217
Form Functions. 217
FrmAlert 217
FrmCloseAllForms 217

FrmCopyLabel 218
FrmCopyTitle 219
FrmCustomAlert 219
FrmDeleteForm 220
FrmDispatchEvent 220
FrmDoDialog 221
FrmDrawForm. 221
FrmEraseForm 222
FrmGetActiveForm 222
FrmGetActiveFormID. 222
FrmGetControlGroupSelection 223
FrmGetControlValue 223
FrmGetFirstForm. 224
FrmGetFocus 224
FrmGetFormBounds 224
FrmGetFormId. 225
FrmGetFormPtr 225
FrmGetGadgetData. 226
FrmGetLabel 226
FrmGetNumberOfObjects 227
FrmGetObjectBounds 227
FrmGetObjectId 228
FrmGetObjectIndex. 228
FrmGetObjectPositon 229
FrmGetObjectPtr 229
FrmGetObjectType 230
FrmGetTitle 230
FrmGetUserModifiedState. 231
FrmGetWindowHandle 231
FrmGotoForm 232
FrmHandleEvent. 232
FrmHelp 233
FrmHideObject 233
FrmInitForm. 234
FrmPopupForm 234
FrmReturnToForm 235

FrmSaveAllForms 235
FrmSetActiveForm 236
FrmSetCategoryLabel 236
FrmSetControlGroupSelection 237
FrmSetControlValue 237
FrmSetEventHandler 238
FrmSetFocus 238
FrmSetGadgetData 239
FrmSetNotUserModified 239
FrmSetObjectPositon 240
FrmSetTitle 240
FrmShowObject 241
FrmUpdateScrollers 241
FrmUpdateForm 242
FrmVisible 242
List UI Functions 243
LstDrawList 243
LstEraseList 243
LstGetNumberOfItems 244
LstGetSelection 244
LstGetSelectionText 245
LstHandleEvent 246
LstMakeItemVisible 247
LstPopupList 247
LstSetDrawFunction 248
LstSetHeight 248
LstSetListChoices 249
LstSetPosition 249
LstSetSelection 250
LstSetTopItem 250
Menu Functions 251
MenuDispose 251
MenuDrawMenu 251
MenuEraseStatus 252
MenuGetActiveMenu 252
MenuHandleEvent 253

MenuInit 254
MenuSetActiveMenu 254

9 Table Functions 255
TblDrawTable 255
TblEditing. 255
TblEraseTable 256
TblFindRowData 256
TblFindRowID 257
TblGetBounds 257
TblGetColumnSpacing 258
TblGetColumnWidth 258
TblGetCurrentField. 259
TblGetItemBounds 259
TblGetItemInt 260
TblGetLastUsableRow 260
TblGetNumberOfRows 260
TblGetRowData 261
TblGetRowHeight 261
TblGetRowID 261
TblGetSelection 262
TblGrabFocus 262
TblHandleEvent 263
TblInsertRow 263
TblMarkRowInvalid 264
TblMarkTableInvalid 264
TblRedrawTable 265
TblReleaseFocus 265
TblRemoveRow 266
TblRowInvalid. 266
TblRowSelectable 267
TblRowUsable 267
TblSelectItem 268
TblSetColumnSpacing 268
TblSetColumnUsable 269
TblSetColumnWidth 269

TblSetCustomDrawProcedure 270
TblSetItemInt 271
TblSetItemPtr 271
TblSetItemStyle 272
TblSetLoadDataProcedure 273
TblSetRowData 274
TblSetRowHeight 274
TblSetRowID 275
TblSetRowSelectable 275
TblSetRowUsable 276
TblSetSaveDataProcedure 277
TblUnhighlightSelection 277

10 Window Functions 279

WinAddWindow 279
WinClipRectangle 279
WinCopyRectangle 280
WinCreateWindow 281
WinCreateOffscreenWindow 282
WinDeleteWindow 283
WinDisableWindow 283
WinDisplayToWindowPt 284
WinDrawBitmap 284
WinDrawChars 285
WinDrawGrayLine 285
WinDrawGrayRectangleFrame 286
WinDrawInvertedChars 286
WinDrawLine 287
WinDrawRectangle 287
WinDrawRectangleFrame 288
WinDrawWindowFrame 288
WinEnableWindow 289
WinEraseChars 289
WinEraseLine 290
WinEraseRectangle 290
WinEraseRectangleFrame 291

WinEraseWindow 291
WinFillLine 292
WinFillRectangle 292
WinGetActiveWindow 293
WinGetClip 293
WinGetDisplayExtent 293
WinGetDisplayWindow 294
WinGetDrawWindow 294
WinGetFirstWindow 295
WinGetFramesRectangle 295
WinGetPattern 296
WinGetWindowBounds 296
WinGetWindowExtent 297
WinGetWindowFrameRect 297
WinGetWindowPointer 298
WinInitializeWindow 298
WinInvertChars 299
WinInvertLine 299
WinInvertRectangle 300
WinInvertRectangleFrame 300
WinModal. 301
WinRemoveWindow 301
WinResetClip 301
WinRestoreBits. 302
WinSaveBits 302
WinScrollRectangle 303
WinSetActiveWindow 304
WinSetClip 304
WinSetDrawWindow 305
WinSetPattern 305
WinSetUnderlineMode 306
WinWindowToDisplayPt 306

11 Miscellaneous User Interface Functions 307
Category Functions 307
CategoryCreateList 307

CategoryEdit 308
CategoryFind 308
CategoryFreeList 309
CategoryGetName 309
CategoryGetNext. 310
CategoryTruncateName 310
CategorySetTriggerLabel 310
CategorySelect 311
Character Attribute Functions 312
GetCharAttr 312
GetCharCaselessValue 312
GetCharSortValue 313
ClipBoard Functions 313
ClipboardAddItem 313
ClipboardGetItem 314
Font Functions 314
FntAccentHeight 314
FntAscent 314
FntAverageCharWidth 315
FntBaseLine 315
FntCharHeight. 315
FntCharsInWidth. 316
FntCharsWidth 317
FntCharWidth 317
FntDescenderHeight 317
FntGetFont 318
FntGetFontPtr 318
FntLineHeight 318
FntLineWidth 319
FntProportionalFont 319
FntSetFont 319
Other User Interface Functions 320
AbtShowAbout 320
DayHandleEvent. 320
Functions for System Use Only 320
Find 320

FindDrawHeader 321
FindGetLineBounds 321
FindSaveMatch 321
FindStrInStr 321
UIInitialize 322
UIReset 322

12 System, Error, Preferences, and Find Functions 323

System Functions 323
SysAppLaunch 323
SysBatteryInfo 324
SysBroadcastActionCode 325
SysCopyStringResource 325
SysCurAppDatabase 326
SysFormPointerArrayToStrings 326
SysHandleEvent 327
SysInsertionSort 328
SysKeyboardDialog 329
SysQSort 329
SysRandom 330
SysReset 331
SysSetAutoOffTime 331
SysTaskDelay 332
SysUIAppSwitch 332
Functions for System Use Only 333
SysAppExit 333
SysAppInfoPtr 333
SysAppInfoPtr SysCurAppInfoP (void) 333
SysAppStartup 333
SysBatteryDialog 333
SysCardImageDeleted 333
SysCardImageInfo 334
SysColdBoot 334
SysCurAppInfoP. 334
SysDisableInts 334
SysDoze 334

SysGetTrapAddress 335
SysInit 335
SysKernelInfo 335
SysLaunchConsole 335
SysLibFind 335
SysLibInstall 336
SysLibRemove. 336
SysLibTblEntry 336
SysNewOwnerID 336
SysPowerOn 336
SysRestoreStatus. 337
SysSetA5 337
SysSetTrapAddress. 337
SysSleep 337
SysUILaunch 337
Error Manager Functions 338
ErrDisplay 338
ErrDisplayFileLineMsg 338
ErrFatalDisplayIf. 339
ErrNonFatalDisplayIf 340
ErrThrow 340
System Preferences Functions 341
PrefGetAppPreferences 341
PrefGetPreferences 342
PrefOpenPreferenceDB 342
PrefSetAppPreferences 343
PrefSetPreferences 343
Find Functions 344
FindDrawHeader 344
FindGetLineBounds 344
FindSaveMatch 345
FindStrInStr 346

13 System Event Manager Functions 347
EvtAddEventToQueue 347
EvtCopyEvent 347

EvtDequeuePenPoint 348
EvtDequeuePenStrokeInfo. 348
EvtEnableGraffiti. 349
EvtEnqueueKey 349
EvtFlushKeyQueue. 350
EvtFlushNextPenStroke 350
EvtFlushPenQueue 351
EvtGetEvent. 351
EvtGetPen. 352
EvtGetPenBtnList 352
EvtKeyQueueEmpty 353
EvtKeyQueueSize 353
EvtPenQueueSize 353
EvtProcessSoftKeyStroke 354
EvtResetAutoOffTimer 354
EvtWakeup 355
Functions for System Use Only 355
EvtDequeueKeyEvent 355
EvtEnqueuePenPoint. 355
EvtGetSysEvent 355
EvtInitialize. 356
EvtSetKeyQueuePtr 356
EvtSetPenQueuePtr 356
EvtSysInit. 356

14 Feature, Time, Float, and String Functions. 357

FtrGet 357
FtrGetByIndex 358
FtrSet 359
FtrUnregister 359
For System Use Only 360
FtrInit 360
String Manager Functions 360
StrAToI 360
StrCat. 360
StrCaselessCompare 361

StrChr . . . 361
StrCompare . . . 362
StrCopy . . . 362
StrIToA . . . 363
StrIToH . . . 363
StrLen . . . 363
StrStr . . . 364
StrToLower . . . 364
Time Manager Functions . . . 365
DateAdjust . . . 365
DateDaysToDate . . . 365
DateSecondsToDate . . . 366
DateToAscii . . . 366
DateToDays . . . 367
DateToDOWDMFormat . . . 367
DayOfMonth . . . 368
DayOfWeek . . . 368
DaysInMonth . . . 368
SelectDay . . . 369
TimAdjust . . . 369
TimDateTimeToSeconds . . . 370
TimGetSeconds . . . 370
TimGetTicks . . . 370
TimSecondsToDateTime . . . 371
TimSetSeconds . . . 371
TimeToAscii . . . 372
Functions for System Use Only . . . 372
TimGetAlarm . . . 372
TimHandleInterrupt . . . 372
TimInit . . . 373
TimSetAlarm . . . 373
Float Manager Functions . . . 373
FplAdd . . . 373
FplAToF . . . 374
FplBase10Info . . . 374
FplDiv . . . 375

FplFloatToLong 375
FplFloatToULong 375
FplFree 376
FplFToA 376
FplInit 377
FplLongToFloat 377
FplMul 377
FplSub 378
Alarm Manager API. 378
AlmGetAlarm 378
AlmSetAlarm 379
Functions for System Use Only. 380
AlmAlarmCallback 380
AlmCancelAll 380
AlmDisplayAlarm 380
AlmEnableNotification 380
AlmInit. 380
Sound Manager Functions 381
SndDoCmd 381
SndGetDefaultVolume 382
SndPlaySystemSound. 382
SndSetDefaultVolume. 383
Functions for System Use Only. 383
SndInit 383

15 Pen, Key, and Graffiti Functions 385
Pen Manager Functions 385
PenCalibrate. 385
PenResetCalibration 386
Functions for System Use Only. 386
PenClose 386
PenGetRawPen 386
PenOpen 387
PenSleep 387
PenRawToScreen 387
PenScreenToRaw 387

PenWake 387
Key Manager Functions 388
KeyCurrentState 388
KeyRates 389
Functions for System Use Only. 389
KeyBootKeys 389
KeyHandleInterrupt 390
KeyInit 390
KeyResetDoubleTap 390
KeySleep 390
KeyWake 390
Graffiti Manager Functions 391
GrfAddMacro 391
GrfAddPoint 391
GrfCleanState 392
GrfDeleteMacro 392
GrfFindBranch. 392
GrfFilterPoints 393
GrfFlushPoints. 393
GrfGetAndExpandMacro 394
GrfGetGlyphMapping 395
GrfGetMacro 395
GrfGetMacroName 396
GrfGetNumPoints 396
GrfGetPoint 396
GrfGetState 397
GrfInitState 397
GrfMatch 398
GrfMatchGlyph 399
GrfProcessStroke 399
GrfSetState 400
SysShortCutListDialog 400
Functions for System Use Only. 400
GrfFieldChange 400
GrfFree 401
GraffitiShift Functions. 401

GsiEnable 401
GsiEnabled 401
GsiInitialize 402
GsiSetLocation 402
GsiSetShiftState 402

1

Developing Palm OS Applications

This chapter helps you understand the basic principles of Palm OS application development. It discusses these topics:

- Overview of Application Development explains the steps involved in creating an application.
- Internal Structure of an Application provides some information about resources the system creates for each application, and how they are used.
- Naming Conventions briefly explains naming conventions used for functions and structures.
- Basic Hardware gives some background information about the Palm OS device.

Overview of Application Development

This section provides an overview of the process of developing an application for the Palm OS device. It introduces the different components of an application in the order that you'll most likely work with them and provides many links to the related sections in this guide and pointers to other relevant documentation included in your developer package.

You learn about these topics:

- Designing UI and Program Functionality
- Constructing UI Resources
- Using Managers and Filling out the Program Logic
- Building, Debugging, and Testing
- Building the Application and Running it on the Device
- Using Other Components of the SDK

Designing UI and Program Functionality

The first step in application development is to envision what users will do as they interact with your application. After that, it's useful to implement a small prototype and have some users interact with it. When you're satisfied with the basic interface and user interaction, you can move on from the prototype to a complete application.

This section looks as the steps involved in creating a working user interface.

Designing Screen Layout and User Interaction

Careful UI design is critical for a Palm OS application because using a Palm OS device differs from using other computers. Here are a few points you have to consider when designing your application:

- Because of the limited screen size, you have to design the layout carefully. Strive for a balance between providing enough information to the user and overcrowding the screen.
- Because users interact with the application using the pen, user input paradigms need to be different from those of keyboard-based applications.
- Limit data input where possible. It often makes sense to let users do the bulk of data input on the desktop while discouraging reliance on Graffiti and the popup keyboard.
- Consider offloading some of the computationally intensive tasks to the desktop; use the device mostly as a satellite viewer if you can.
- Plan integration with the desktop early. Your conduit/backup strategy and you integration with desktop software can greatly enhance the usefulness of your program.

The Palm OS development team has developed a set of design guidelines that were used as the basis for the four applications resident on the device (Note Pad, Address Book, etc.). These guidelines are summarized in Chapter 5 of the "Palm OS Cookbook." Some information, such as recommended font size or border width, is included in Chapter 3, "Palm OS User Interface Resources."

Follow the design guidelines in chapters 4 and 5 of the "Palm OS Cookbook" to make your application easier to learn and to use.

Constructing UI Resources

The ResEdit resource templates that were used to implement all the applications resident on the device are provided with your development environment. You can use ResEdit with the resource templates to create your own buttons, pop-up lists, menus, and other parts of the user interface.

The process of creating new resources is described in detail in the tutorial; the basic process consists of entering values into the attribute fields of the resource templates. Each resource has to have an ID and may also need a width, height, label, or other attributes. The recommended (or required) values for the different fields in each resource are provided in Chapter 3, "Palm OS User Interface Resources."The "Palm OS Tutorial" provides "recipes" for creating each resource type in the Tutorial Cookbook chapter.

When you build your program, the system converts the ResEdit resources into data structures that the system can work with. Different resource types map to a different data structures, that is, UI object type. For example, menu resources map to objects that know how to turn highlighting on and of and fields know how to position input cursors and process user input. The operating system provides quite a bit of default functionality for each UI object type. Your program logic can use, replace, or extend that functionality. Detailed information on all structures and their fields is provided in Chapter 5, "Palm OS UI Objects."

Using Managers and Filling out the Program Logic

To successfully build a Palm OS application, you need to understand how the system itself is structured and how to structure your application.

Using Events and Launch Codes

Palm OS applications are single-threaded event-driven programs. The events are generated by the system based on user inputs and system interrupts. The program logic may generate events as well. The programs are structured as a series of event handlers dispatched from a single event loop in each program.

In addition, a number of launch codes regulate how applications interact with each other; for example, one application can invoke a find action on another using a launch code. This is described in more detail in How Action Codes Control the Application.

NOTE: To make your application interact appropriately with other applications on the device, and to avoid other problems later, read Chapter 4, “Design, Testing, and Localization Guidelines” of the Palm OS Cookbook.

Using the Palm OS Managers

The Palm OS system API is divided into functional areas called managers. Each manager has a distinct three-letter prefix used on all API calls and structures and is discussed separately below.

- The system manager, system event manager, feature manager, float manager, string manager, and so on are discussed in Chapter 6, “Using Palm OS Managers.”
- The memory manager, data manager, and resource manager are explained in Chapter 1, “Palm OS Memory Management” of “Developing Palm OS Applications, Part II.”
- The communications API is explained in Chapter 2, “Palm OS Communications” of “Developing Palm OS Applications, Part II.”

NOTE: Avoid using functions from standard desktop C libraries. These will significantly slow down and enlarge your program. Many will not work at all on the device. Use functions provided by the Palm OS managers instead.

Building, Debugging, and Testing

To build your application for initial debugging and testing, you use the CodeWarrior IDE and the Pilot Simulator library, which lets you build and debug applications on the Macintosh. Documentation for the CodeWarrior IDE and source-level debugger is provided with CodeWarrior.

After you've built the application, you can use the Pilot Simulator to run it on a simulated Pilot device on the Macintosh screen. You can interact with the simulated buttons, menus, or fields, and even enter Graffiti characters using the mouse. You can also use the Simulator to test your application using an automated test suite called Gremlins. Using the Simulator is discussed in detail in chapter 2, "Using the Palm OS Simulator," of the "Palm OS Cookbook."

Building the Application and Running it on the Device

When you've completed building and testing the application with the Simulator, you can compile it with a Makefile from MPW and download and run the executable on the device. This process is described in Chapter 1, "Running an Application on the Palm OS Device," and Chapter 3, "File Transfer With the Pilot Debugger" of the Palm OS Cookbook. Phase 20 of the Palm OS Tutorial provides step-by-step instructions for changing a Makefile, compiling an application, downloading the application onto the device, and running it.

Using Other Components of the SDK

The Palm OS has provided the following additional items in the development kit to help you come up to speed quickly:

- The Palm OS tutorial provides step-by-step examples of developing an application from start to finish in its twenty phases. Examples, both resources and code that is incrementally changed, are included.
- The actual source code for the four PIM applications on the Pilot is included as examples. The code can be a valuable aid when you develop your own program. The software development kit provides a royalty-free license that permits you to use any or all of the source code from the examples in your application.

Internal Structure of an Application

Every application running under Palm OS must have certain minimum system (not UI) resources defined to be recognized by the Palm OS system software. These required resources are created for you by the development environment. Most applications have other resources that are application specific in addition to the required minimum resources. (Throughout this chapter, resources are referenced by type and ID where the type is a 4 byte ascii string like 'code' and the ID is a decimal integer.)

The system creates a 'code' #1 resource for every application. This resource is the entry point for the application and is where application initialization is performed. When the Palm OS device launches an application, it starts executing at the first byte of the 'code' #1 resource. All of the application code that you provide is included in this resource as well.

Typically, this is startup code provided with the Palm OS development environment that is linked in with your application code. This startup code works as follows:

- The startup code performs application setup and initialization
- The startup code calls your "main" routine
- When your "main" routine exits, control is returned to the startup code which performs any necessary cleanup of your application and returns control to the Palm OS system software.

The system also creates a 'pref' #0 resource for every application. This resource contains startup information necessary for launching your application. The resource includes the required stack size, the dynamic heap space required, and the task priority. Note that although the 'pref' #0 resource must be present, it is mainly for future use since in the current version of the system user-interface applications do not get their own stack or priority.

The final required resources are the 'code' #0 and 'data' #0 resources. These resources contain the required size of your global data and an image of the initialized area of that global data. When your application is launched, a memory chunk in the dynamic heap is allocated which is big enough to hold all of your globals and then the 'data' #0 resource is used to initialize those globals.

Naming Conventions

The following conventions are use throughout the Palm OS API:

- Functions start with a capital letter.
- All functions belonging to a particular manager start with a two- or three-letter prefix, such as "Ctl" for control functions or "Ftr" for functions that are part of the feature manager.
- Events and other constants start with a lower case letter.
- Structure elements start with a lower case letter.
- Global variables start with a capital letter.
- Typedefs start with a capital letter and end with "type" (for example, `DateFormatType`, found in DateTime.h).
- Resource types usually start with a lower case followed by three capital letters, for example tSTR or tTBL. (There are some customized Macintosh resources provided with your developer package. They are all upper case, for example, MENU.)
- Members of an enumerated type start with a lower-case prefix followed by a name starting with a capital letter, as follows:

```
enum formObjects {
  frmFieldObj,
  frmControlObj,
  frmListObj,
  frmTableObj,
  frmBitmapObj,
  frmLineObj,
  frmFrameObj,
  frmRectangleObj,
  frmLabelObj,
  frmTitleObj,
  frmPopupObj,
  frmGraffitiStateObj,
  frmGadgetObj};
typedef enum formObjects FormObjectKind;
```

Basic Hardware

This section helps you understand the device you're developing your application for by discussing RAM and ROM, Palm OS Modes of Operation, Palm OS Connectivity, Real-Time Clock and Timer, Palm OS Device Screen and Sound Generation, Palm OS Device Screen and Sound Generation, and Palm OS Device Reset Switch.

RAM and ROM

The first version of Palm OS runs on the Motorola 68328 "Dragon-Ball" processor. The first memory card shipped with the device has 128K of pseudo-static RAM and 512K of ROM for the system software and application code. A portion of the RAM (32K) is reserved for system use and is not available for storing user data. Both the ROM and RAM are on a memory module which users can replace. The Palm OS device does not have a disk drive or PCMCIA support.

Palm OS Modes of Operation

To minimize power consumption, Palm OS dynamically switches between 3 different modes of operation: sleep mode, doze mode, and running mode.

- In *sleep mode*, the device looks like it is turned off: the display is blank, the digitizer is inactive, and the main clock is stopped. The only circuits still active are the real-time clock and interrupt generation circuitry.

  The device enters this mode when there is no user activity for a number of minutes or when the user presses the "off" button. The device only comes out of sleep mode when there is an interrupt, for example, when the user presses a button.
- In *doze mode*, the main clock is running, the device appears like it is turned on, and the processor's clock is running but it is not executing instructions (that is, it is halted). When the processor gets an interrupt, it comes out of halt and starts processing the interrupt.

  The device enters this mode it's on but has no user input to process.
- In *running mode*, the processor is actually executing instructions.

The device enters this mode when it detects user input (like a tap on the screen) while in doze mode, or when it detects an interrupt while in doze or sleep mode. The device stays in running mode only as long as it takes to process the user input (most likely less than a second) then it immediately re-enters doze mode.

To maximize battery life, the processor on the Palm OS device is kept out of running mode as much as possible. Any interrupt generated on the device must therefore be capable of "waking" up the processor. The processor can receive interrupts from the serial port, the hard buttons on the case, the button on the cradle, the programmable timer, the memory module slot, the real-time clock (for alarms), the low battery detector, and any built-in peripherals such as a pager or modem.

Palm OS Connectivity

The Palm OS device uses its serial port for implementing desktop PC connectivity or other external communication. The serial communication is fully interrupt-driven for receiving data. Currently, interrupt-driven transmission of data is not implemented in software, but the hardware does support it. There are 5 external signals used for this communication: signal ground (SG), transmit data (TxD), receive data (RxD), clear to send (CTS), and request to send (RTS).

The Palm OS device has an external connector that provides the 5 serial communication signals, a general purpose output, a general purpose input, and a cradle button input.

Real-Time Clock and Timer

The Palm OS device has a real-time clock and programmable timer as part of the 68328. The real time clock maintains the current time even when the system is in sleep mode (turned "off"). It is capable of generating an interrupt to wake the device when an alarm is set by the user. The programmable timer is used to generate the system tick count interrupts (100 times/sec.) while the processor is in doze or running mode. The system tick interrupts are required for periodic activity such as polling the digitizer for user input, key debouncing, etc.

The Palm OS device has one memory module socket for installing modules which may contain ROM or RAM storage.

Palm OS Device Screen and Sound Generation

The first version of the Palm OS device has an LCD screen of 160x160 pixels. The LCD controller built into the 68328 maps a portion of system memory to the LCD. Currently, the software only supports 1 bit/pixel monochrome graphics although the controller can support 2 bits/pixel gray scale.

The Palm OS device has a built-in digitizer overlaid onto the LCD screen and extending about an inch below the screen. This digitizer is capable of sampling accurately to within 0.35 mm (.0138 in) with up to 50 accurate points/second. When the device is in doze mode, an interrupt is generated when the pen is first brought down on the screen. After a pen down is detected, the system software polls the pen location periodically (every 20ms) until the pen is again raised.

The Palm OS device has primitive sound generation. A square wave is generated directly from the 68328's PWM circuitry. There is frequency and duration control but no volume control.

Palm OS Device Reset Switch

The Palm OS device has a reset button for resetting the processor and forcing a boot-up sequence:

- Simply pressing the reset switch causes a soft reset which does not destroy any user data.
- Holding down the power button while pressing the reset switch causes a hard reset which erases all user data following a confirmation by the user.

2

Application Control Flow

Palm OS applications are generally single-threaded, event-driven programs. They may use predefined UI elements (sometimes referred to as UI objects) or they may create their own. All applications must use the memory and data management facilities provided by the system and must be considerate of the system and other applications by periodically allowing system event handlers access to the event flow.

The flow of control in Palm OS is driven by two different mechanisms, discussed in some detail in this chapter:

- How Events Control an Application discusses the event manager, the main interface between the Palm OS system software and an application. It discusses in some detail what an application does in response to user input, providing code fragments as examples where needed.
- How Action Codes Control the Application discusses how an application handles requests for immediate action at its top level (`PilotMain`). For example, there are action codes for launching an application, for telling an application to search its data for a text string, and for notifying an application that data has been synchronized. Using action codes, an application can request information or actions from another application.

How Events Control an Application

This section starts with a high-level overview of the stages of a Palm OS application, then discusses the event loop in some detail.

Note that each event is discussed in some detail in Chapter 4, "Application Control Flow."The event flow for each User Interface resource is discussed in Chapter 3, "Palm OS User Interface Resources."The event flow for each User Interface object is discussed in Chapter 5, "Application Control Flow."

Basic Application Stages

When an application receives a the action code `sysAppLauchCommandNormalLaunch` (see How Action Codes Control the Application), it begins with a startup routine, then goes into an event loop, and finally exits with a stop routine.

- The Startup Routine is the application's opportunity to perform actions which need to happen once, and only once, at startup. A typical startup routine opens databases, reads saved state information (such as UI preferences) and initializes the application's global data.
- The Event Loop fetches events from the queue and dispatches them, taking advantage of default system functionality as appropriate.
- The Stop Routine is the application's opportunity to perform cleanup activities before exiting. Typical activities include closing databases and saving state information.

The following sections look at each of the phases in some detail. Note that for each phase, Palm OS provides a default behavior that can help you keep application code at a minimum. If your application has special requirements, your application may instead handle the bulk of the work itself.

The Startup Routine

During the startup routine, an application has to follow these steps:

1. Get system-wide preferences (for example for numeric or date and time formats) and use them to initialize global variables that will be referenced throughout the application.
2. Find the application database by creator type. If none exists, create it and initialize it.
3. Get application-specific preferences and initialize related global variables.
4. Initialize any other global variables.

Listing 2.1 shows an example `StartApplication` function from the datebook application.

Listing 2.1 StartApplication from Datebook.c

```
static Word StartApplication (void)
{
  Word error = 0;
  Err err = 0;
  UInt mode;
  DateTimeType dateTime;
  DatebookPreferenceType prefs;
  SystemPreferencesType sysPrefs;

  // Determine if secret record should be shown.
  PrefGetPreferences (&sysPrefs);
  HideSecretRecords = sysPrefs.hideSecretRecords;

  if (HideSecretRecords)
    mode = dmModeReadWrite;
  else
    mode = dmModeReadWrite | dmModeShowSecret;

  // Get the time formats from the system 
preferences.
  TimeFormat = sysPrefs.timeFormat;

  // Get date formats from system preferences.
  LongDateFormat = sysPrefs.longDateFormat;
  ShortDateFormat = sysPrefs.dateFormat;

  // Get start day of week from system preferences.
  StartDayOfWeek = sysPrefs.weekStartDay;

  // Get today's date.
  TimSecondsToDateTime
          (TimGetSeconds (), &dateTime);
  Date.year = dateTime.year - firstYear;
  Date.month = dateTime.month;
  Date.day = dateTime.day;
```

```
    // Find application's data file. Create one if
    // none exists.
    ApptDB DmOpenDatabaseByTypeCreator
          (datebookDBType, sysFileCDatebook, mode);
    if (! ApptDB)
      {
      error = DmCreateDatabase (0, datebookDBName,
          sysFileCDatebook, datebookDBType, false);
      if (error) return error;

      ApptDB = DmOpenDatabaseByTypeCreator
          (datebookDBType, sysFileCDatebook, mode);
      if (! ApptDB) return (1);

      error = ApptAppInfoInit (ApptDB);
      if (error) return error;
      }


    // Read preferences & saved-state information.
    if (PrefGetAppPreferences (sysFileCDatebook,
             datebookVersionNum, &prefs,
            sizeof (DatebookPreferenceType)))
      {
      DayStartHour = prefs.dayStartHour;
      DayEndHour = prefs.dayEndHour;
      AlarmPreset = prefs.alarmPreset;
      NoteFont = prefs.noteFont;
      SaveBackup = prefs.saveBackup;
      }

    TopVisibleAppt = 0;
    CurrentRecord = noRecordSelected;

    return (error);
  }
```

The Event Loop

When startup is complete, the application enters an event loop. It typically remains in that event loop until the system tells it to shut itself down by sending an `appStopEvent` (not a action code).

Listing 2.2 Top-level event loop example

```
static void EventLoop (void)
{
  Word error;
  EventType event;

  do
    {
    EvtGetEvent (&event, evtWaitForever);

    if (! SysHandleEvent (&event))

      if (! MenuHandleEvent (NULL, &event, &error))

        if (! ApplicationHandleEvent (&event))

          FrmDispatchEvent (&event);

    }
  while (event.eType != appStopEvent);
}
```

In the event loop, the application iterates through these steps (see Figure 2.1)

1. Fetch an event from the event queue.
2. Call `SysHandleEvent` to give the system an opportunity to handle the event.

   The system handles events like power on/ power off, Graffiti input, tapping silk-screened icons, or pressing buttons. During the call to `SysHandleEvent`, the user may also be informed about low-battery warnings or may find and search another application.

Note that in the process of handling an event, `SysHandleEvent` may generate new events and put them on the queue. For example, the system handles Graffiti input by translating the pen events to key events. Those, in turn, are put on the event queue and are eventually handled by the application.

`SysHandleEvent` returns TRUE if the event was completely handled, that is, no further processing of the event is required. The application can then pick up the next event from the queue.

3. If `SysHandleEvent` did not completely handle the event, the application calls `MenuHandleEvent`. `MenuHandleEvent` handles two types of events:
   – If the user has tapped in the area that invokes a menu, `MenuHandleEvent` brings up the menu.
   – If the user had tapped inside a menu to invoke a menu command, `MenuHandleEvent` removes the menu from the screen and puts the events that result from the command onto the event queue.

   `MenuHandleEvent` returns TRUE if the event was completely handled.
4. If `MenuHandleEvent` did not completely handle the event, the application calls `ApplicationHandleEvent`. `ApplicationHandleEvent` handles only the `frmLoadEvent` for that event; it loads and activates application form resources and sets the event handler for the active form.
5. If `ApplicationHandleEvent` did not completely handle the event, the application calls `FrmDispatchEvent`. `FrmDispatchEvent` first sends the event to the application's event handler for the active form. This is the event handler routine that was established in `ApplicationHandleEvent`. Thus the application's code is given the first opportunity to process events that pertain to the current form. The application's event handler may completely handle the event and return TRUE to calls `FrmDispatchEvent`. In that case, calls `FrmDispatchEvent` returns to the application's event loop. Otherwise, calls `FrmDispatchEvent` calls `FrmHandleEvent` to provide the system's default processing for the event.

For example, in the process of handling an event, an application frequently has to first close the current form and then open another one, as follows:

- The application calls `FrmGotoForm` to bring up another form. `FrmGotoForm` queues a `frmCloseEvent` for the currently active form, then queues `frmLoadEvent` and `frmOpenEvent` for the new form.
- When the application gets the `frmCloseEvent`, it closes and erases the currently active form.
- When the application gets the `frmLoadEvent`, it loads and then activates the new form. Normally, the form remains active until it is closes. (Note that this wouldn't work if you preload all forms, but that's really discouraged. Applications don't need to be concerned with the overhead of loading forms, it's fast enough so they can do it when they need it.) The application's event handler for the new form is also established.
- When the application gets the `frmOpenEvent`, it does whatever initialization of the form is required, then draws the form on the display.

After `FrmGotoForm` has been called, any further events that come though the main event loop and to `FrmDispatchEvent` are dispatched to the event handler for the form that is currently active. The event handler knows for a particular dialog box or form how it should respond to events for example, opening, closing, and so on.

`FrmHandleEvent` invokes the default UI functionality.

After the system has done all it can to handle the event for the specified form, the application finally calls the active form's own event handling function; for example, in the datebook application, it may call `DayViewHandleEvent` or `WeekViewHandleEvent`.

Note again how the structure of the event flow allows your application to rely on system functionality as much as it wants. If your application wants to know whether a button is pressed, it has to only wait for `ctlSelectEvent`. All the details of the event queue are handled by the system.

Some events are actually requests for the application to do something. For example, `frmOpenEvent`. Typically, all the application does is draw itself using the functions provided by the system and then waits for events it can handle to arrive from the queue.

Only the active form should process events.

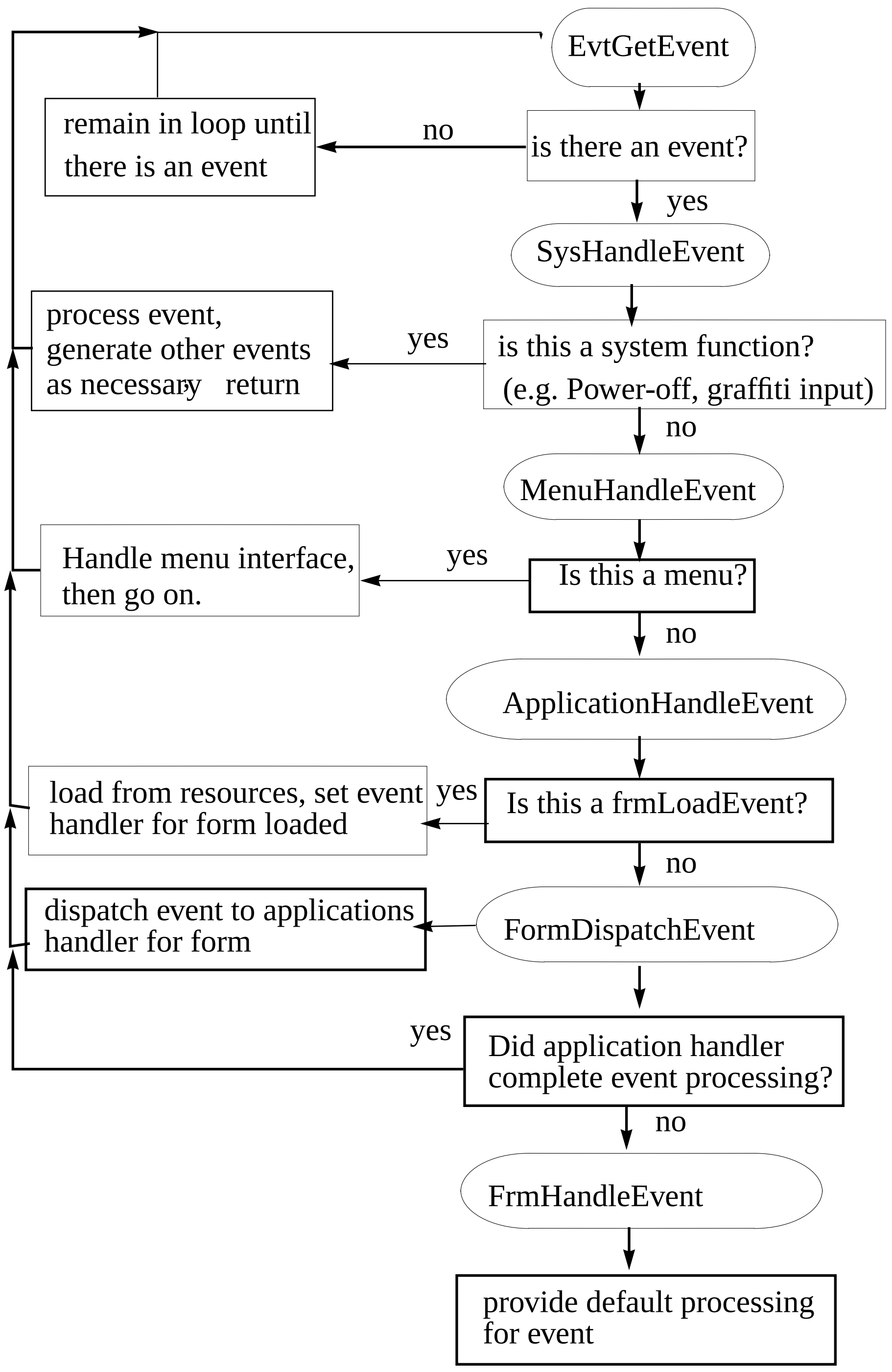
The Stop Routine

The stop routine should first flush all active records and then close the applications database and saves those aspects of the current state that are necessary for startup. Listing 2.3 provides an example of a `StopApplication` routine from Datebook.c.

Listing 2.3 Example for StopApplication Routine

```
static void StopApplication (void)
{
  DatebookPreferenceType prefs;

  // Write preferences & saved-state information.
  prefs.noteFont = NoteFont;
  prefs.dayStartHour = DayStartHour;
  prefs.dayEndHour = DayEndHour;
  prefs.alarmPreset = AlarmPreset;
  prefs.saveBackup = SaveBackup;

  // Write the state information.
  PrefSetAppPreferences (sysFileCDatebook,
      datebookVersionNum, &prefs,
      sizeof (DatebookPreferenceType));

  // Send a frmSave event to all open forms.
  FrmSaveAllForms ();

  // Close all the open forms.
  FrmCloseAllForms ();

  // Close the application's data file.
  DmCloseDatabase (ApptDB);
}
```

Figure 2.1 Control Flow in a Typical Application


EvtGetEvent
remain in loop until there is an event
no
is there an event?
yes
SysHandleEvent
process event, generate other events as necessary return
yes
is this a system function? (e.g. Power-off, graffiti input)
no
MenuHandleEvent
Handle menu interface, then go on.
yes
Is this a menu?
no
ApplicationHandleEvent
load from resources, set event handler for form loaded
yes
Is this a frmLoadEvent?
no
dispatch event to applications handler for form
FormDispatchEvent
yes
Did application handler complete event processing?
no
FrmHandleEvent
provide default processing for event

How Action Codes Control the Application

Action codes provide a direct communication mechanism between the system and an application or between two applications:

- Inter-application communication is implemented through the action code mechanism. An application can use an action code to request that another application modify its data or perform an action. For example, a data collection application could instruct an email application to queue up a particular message to be sent.
- The system uses action codes to ask an application to do something (interrupting other activities if necessary). Examples are action codes for launching an application, initializing databases, or hard reset. Another example is the global find. When the user performs a global find, it would be wasteful to incur the overhead of a full launch of each application as its data file is searched. Instead, the system sends an action code with a special flag to the application has the application perform the search without displaying its user interface to other application overhead.

Action codes may be sent to any application without negative effects. However, an action code only has an effect if the application that receives it has been programmed to handle it. When developing your application, you, handle as many of the standard action codes as possible. The standard action codes defined by Palm OS are listed in Table 2.1.

The system delivers action codes to the application at its highest level (`PilotMain`). Each action code is accompanied by a parameter block containing the necessary data. The system sometimes uses flags with an action code that indicate its circumstances, for example, whether the UI is to be displayed. Action codes may be sent from the system's top level or from another application's thread; in most cases, global variables are not available.

Action Code Example

An application needs to checks for action codes in its main function. Listing 2.4 shows an example from the datebook application.

Listing 2.4 Code Fragment Checking for Action Codes

```
DWord PilotMain (Word cmd, Ptr cmdPBP, Word
launchFlags)
{
  Word error;
  Boolean launched;

// Launch code sent by launcher or datebook button.
  if (cmd == sysAppLaunchCmdNormalLaunch)
    {
    error = StartApplication ();
    if (error) return (error);

    FrmGotoForm (DayView);
    EventLoop ();
    StopApplication ();
    }


  // Launch code sent by text search.
  else if (cmd == sysAppLaunchCmdFind)
    {
    Search ((FindParamsPtr)cmdPBP);
    }


// This action code is sent when if user taps GoTo
// button in Find Results dialog box (application
// may be already running)

  else if (cmd == sysAppLaunchCmdGoTo)
    {
    //Determine if this app is already running
    launched = launchFlags
                  sysAppLaunchFlagNewGlobals;
    if (launched)
```

```
      {
      //Not yet running so start it.
      error = StartApplication ();
      if (error) return (error);

      GoToItem ((GoToParamsPtr) cmdPBP, launched);

      EventLoop ();
      StopApplication ();
      }
    else
    //Go to the search destination
      GoToItem ((GoToParamsPtr) cmdPBP, launched);
    }


// Launch code sent by sync application to notify
// application that its database was synced.

  else if (cmd == sysAppLaunchCmdSyncNotify)
    {
    SyncNotification ();
    }

....
```

Responding to Action Codes

When an application receives an action code, it must first check if it can handle it. For example, only applications that have text data should respond to an action code requesting a string search. If an application can't handle an action code, it exits without failure. Otherwise, it performs the action immediately and returns.

Predefined Action Codes

A number of action codes are predefined by the system for handling certain system tasks, for example,

- notifying the application when certain system preferences like date and time have changed
- performing global find and goto operations
- notifying the application that its data files have been updated by a sync operation

The action code parameter is a 16-bit word value. All action codes with values 0-32767 are reserved for use by the system and for future enhancements. Action codes 32768 - 65535 are available for private use by applications.

Table 2.1 Palm OS Action Codes

Code	Request
sysAppLaunchCmdNormalLaunch	Normal launch
sysAppLaunchCmdFind	Find text string
sysAppLaunchCmdGoTo	Launch and go to a particular record and optionally select the find text
sysAppLaunchCmdSyncNotify	Sent to applications whose databases changed during HotSync after the sync has been completed
sysAppLaunchCmdTimeChange	System time changed
sysAppLaunchCmdSystemReset	Sent after System hard resets
sysAppLaunchCmdAlarmTriggered	Schedule next alarm
sysAppLaunchCmdDisplayAlarm	Display given alarm dialog
sysAppLaunchCmdCountryChange	Country has changed
sysAppLaunchCmdSyncRequest	HotSync button was pressed

Table 2.1 Palm OS Action Codes

Code	Request
sysAppLaunchCmdSaveData	Sent to running app before action codes that cause data search or manipulation, such as `sysAppLaunchCmdFind`.
sysAppLaunchCmdInitDatabase	Initialize database; sent by DesktopLink server to application whose create ID matches that of the database created in response to the create database request.
sysAppLaunchCmdSyncCallApplication	Used by DesktopLink Server command “call application”

Action Code Flags

When the system sends an action code, it may send flags determining application behavior. Applications should always use zero.

Action Code Parameter Blocks

Some action codes are called in conjunction with a parameter block structure that provides more information about the request. This section provides information about available parameter blocks and the values of their fields for these action codes:

- sysAppLaunchCmdSaveData
- sysAppLaunchCmdSystemReset
- sysAppLaunchCmdInitDatabase
- sysAppLaunchCmdSyncCallApplication
- sysAppLaunchCmdGoto Command
- sysAppLaunchCmdFind

sysAppLaunchCmdSaveData

```
typedef struct {
  Boolean  uiComing;
  } SysAppLaunchCmdSaveDataType;
```

uiComing	True if system dialog is put up before action code arrives.

sysAppLaunchCmdSystemReset

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

hardReset	True if system was hardReset
createDefaultDB	If true, application has to create default database.

sysAppLaunchCmdInitDatabase

```
typedef struct SysAppLaunchCmdInitDatabaseType {
        DmOpenRef          dbP;
        ULong              creator;
        ULong              type;
        UInt               version;
  } SysAppLaunchCmdInitDatabaseType;
```

dbP	Database reference.
creator	Database creator.
type	Database type.
version	Database version.

sysAppLaunchCmdSyncCallApplication

```
typedef struct
SysAppLaunchCmdSyncCallApplicationType {
  Word      action;
  Word      paramSize;
  VoidPtr   paramP;
  Byte      remoteSocket;
  Byte      tid;
  Boolean   handled;
  } SysAppLaunchCmdSyncCallApplicationType;
```

action	Call action id (application specific).
paramSize	Parameter size.
paramP	Pointer to parameter.
remoteSocket	Remote Socket ID
tid	Command transaction
handled	Must be set to TRUE by the application if handled.

sysAppLaunchCmdGoto Command

```
typedef struct {
  Word    searchStrLen;
  Word    dbCardNo;
  LocalID dbID;
  Word    recordNum;
  Word    matchPos;
  Word    matchFieldNum;
  DWord   matchCustom;
  } GoToParamsType;
```

searchStrLen	Length of search string.
dbCardNo	Card number of the database
dbID	LocalID of the database
recordNum;	Index of record that contain a match
matchPos	Position in record of the match.
matchFieldNum	Field number string was found in
matchCustom	Application specific info

sysAppLaunchCmdFind

```
typedef struct {

// These fields are used by the applications.
  Word               dbAccesMode;
  Word               recordNum;
  Boolean            more;
  Char               strAsTyped [maxFindStrLen+1];
  Char               strToFind [maxFindStrLen+1];

// These fields are private to the Find routine
//and should NOT be accessed by applications.
  Word               numMatches;
  Word               lineNumber;
  Boolean            continuation;
  Boolean            searchedCaller;
  LocalID            callerAppDbID;
  Word               callerAppCardNo;
  LocalID            appDbID;
  Word               appCardNo;
  Boolean            newSearch;
  DmSearchStateType  searchState;
  FindMatchType      match [maxFinds];
} FindParamsType;
```

dbAccessMode	read mode and maybe show secret
recordNum	index of last record that contained a match
more	true if more matches to display
strAsTyped [maxFindStrLen+1]	search string as entered
strToFind [maxFindStrLen+1]	search string is lower case
numMatches	System use only.

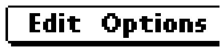
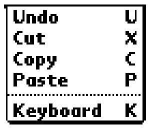
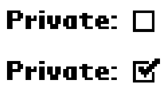
lineNumber	System use only.
continuation	System use only.
searchedCaller	System use only.
callerAppDbID	System use only.
callerAppCardNo	System use only.
appDbID	System use only.
appCardNo	System use only.
newSearch	System use only.
searchState	System use only.
match [maxFinds]	System use only.

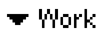
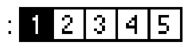
Creating Your Own Action Codes

In addition to the predefined action codes defined in Table 2.1, developers may create their own action codes to implement specific functionality. Both the sending and the receiving application must know about and handle any developer-defined action codes.

Palm OS User Interface Resources

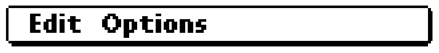
This chapter provides an overview of all Macintosh resources used by Palm OS, followed by a detailed description of each resource. The relationship between the Macintosh resources and the structures provided by Palm OS is discussed in "Palm OS UI Objects".

Resource	Name	UI Name	ResEdit Resource
Menu Bar Resource	MBAR	Menu bar	
Menu Resource	MENU	Menu	
Application Icon Name Resource	tAIN	Application icon name	
Alert Resource	Talt	Alert	
Button Resource	tBTN	Button	
Check Box Resource	tCBX	Check box	
Field Resource	tFLD	Field	
Form Bitmap Resource	tFBM	Form bitmap	
Form Resource	tFRM	Form	

Resource	Name	UI Name	ResEdit Resource
Gadget Resource	tGDT	Gadget	
Graffiti Shift Resource	tGSI	Graffiti® Shift	
Label Resource	tLBL	Label	
List Resource	tLST	List box	
Popup List Resource	tPUL	Popup list	
Popup Trigger Resource	tPUT	Popup trigger	
Push Button Resource	tPBN	Push button	
Repeating Button Resource	tREP	Repeating control	
Selector Trigger Resource	tSLT	Selector trigger	
String Resource	tSTR	String	
Table Resource	tTBL	Table	
Title Resource	tTTL	Title	
Version Number String	tver	Version number string	

Menu Bar Resource

Name	MBAR (This is a Macintosh resource.)
UI Name	Menu bar
Overview	The menu bar UI object groups menus. The menu bar shows all menus, provided there is room for them. Here's a picture of a two-item menu bar, with no menu selected:



Attributes		
	# of menus	Array of N objects.
	Menu res ID	MENU resource ID of the nth menu object.

Comments: If you create more menus than the display can handle, the result is a runtime error. The display can usually handle four menus at most, depending on the number of characters each menu name contains.

To create a menu, you need both a MENU resource and an MBAR resource. The tutorial provides a detailed example for creating a menu.

See Also: MENU resource.

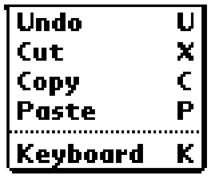
Menu Resource

Name	MENU (This is a Macintosh resource.)
UI Name	Menu

Overview: A menu provides access to commands not available on screen. Tapping the MENU silk-screened icon provides access to the menu commands.

Each menu can have a Graffiti keystroke equivalent: The user draws the command stroke and one key to execute a command. For example, writing Command-X executes the Cut command.

The menu UI object lets you define the popup menus triggered from the menu bar. ResEdit provides a graphical interface for specifying a menu. The following five-command menu has a separator line after the fourth item and shortcuts for each item:



Attributes	Title	String of the menu command.

Comments The Enable check box in the ResEdit dialog has no effect on the Simulator and is ignored. Separator lines and shortcuts appear as in the figure above. The display cannot handle more than thirteen commands in a menu. If you create a menu with more commands, a runtime error results.

The tutorial provides a detailed example for creating a menu.

Event Flow for Menu Resource

User Action	System Response
Pen enters menu window.	`winExitEvent` to exit previous window. `winEnterEvent` to enter menu window. `penDownEvent` is also triggered, although the pen has not actually touched the screen.
User selects a menu item.	`WinExitEvent` to exit menu window. `WinEnterEvent` to enable the form the menu spawned. `menuEvent` (store ID number of the item in `EventType`). `penUpEvent` finally occurs.

Menus and Menu Bars

A menu consists of a menu bar, menu names indicating the available menus, and the menus themselves with their commands:

- **Menu bar**. The menu bar at the top of the screen contains the names of the available menus. Each application has different sets of menu names; within an application, different views may have different menus.
- **Menu name**. Each menu is displayed below the menu name.
  - Record—Place Record to the left of Edit (if applicable).
  - Edit—Screens that allow editing need an Edit menu.
  - Options—Typically the last menu. The About command is always an Options command.
- **Menu**. Menus consist of menu commands and optional shortcuts. Menu commands should not duplicate functionality available via command buttons. Menus justify left with the active heading of the menu name when invoked. If the menu does not fit, it is justified to the right border of the screen.

NOTE: For each menu, provide shortcuts for all commands or for none at all.

A pen-up on the menu icon displays the menu bar. The first time a menu is invoked after an application is launched, no menus are displayed unless there is only one menu available. Afterwards, on a pen-down of the digitizer, the menu and menu item of the last command executed from the menu are displayed (Graffiti command equivalents are ignored). For example, if the user selects Copy from the Edit menu, the menu is popped down and the Copy command is highlighted the next time the menu bar is displayed. This expedites execution of commonly used commands or grouped commands (e.g., Copy/Paste). The last menu heading is not saved if the user switches to a different view or a different application.

Each view within an application can have a unique menu (i.e., different menu headings and items). There are no grayed out menu headings or grayed-out menu items. A command not accessible in a certain mode does not appear at all.

After a menu command is executed, the menu bar is dismissed.

The menu bar is active when the menu headings in it are active. When not active, the menu bar is not visible.

The vertical active area of menu headings is 2 pixels beyond the ascender and 1 pixel below a potential descender of the menu heading text. The horizontal active area covers half of the distance to the next menu heading, leaving no gaps between the headings. If the menu headings aren't as wide as the menu bar, part of it may be inactive.

The entire area of the menu, excluding the border, is active. Divider lines and status items on the launcher menu are inactive; that is, they do not highlight when tapped.

Application Icon Name Resource

Name	tAIN
UI Name	Application icon name
Overview	This resource associates a name with an application icon. The name is displayed by the launcher. This name overrides the name of the application file.

Attributes	App icon name	Name displayed with the application icon, in the launcher.

Alert Resource

Name	Talt
UI Name	Alert
Overview	The alert resource is used to define a modal dialog that displays a message, an icon, and one or more buttons.

Attributes

Alert Type	Determines the sound played and the icon displayed when the alert is drawn. There are four possible icons, informationAlert (Alert Number 0), confirmationAlert (Alert Number 1), warningAlert (Alert Number 2), and ErrorAlert (Alert Number 3).
Help Res ID	Resource ID of a string resource (tSTR) that is the help text for the alert dialog box.
# of Buttons	Number of buttons in the alert form.
Default Button	Default button for the alert form.
Title	Title of the alert form.
Message	Message displayed by the alert dialog. May contain ^1, ^2, ^3 as substitution variables to use in conjunction with `FrmCustomAlert`.
Button Text	Text of the buttons, determined by an entry in the resource of each button. The label "1) *****" is a placeholder for the first button. Select this label and press Command-K to add a new entry.

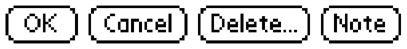
System Alerts

System alerts are text-only dialog boxes that indicate error messages, alarms, etc. Alerts typically have only an OK or an OK and Cancel, and no controls. A small icon should indicate the category of the dialog box; for example, an exclamation mark for an error message. The icon appears on the left side of the dialog. The text is justified left but placed to the right of the dialog icon.

Types	Icon	Definition	Options	Example
Information	i	Lowest-level warning for an action that shouldn't or can't be completed, but doesn't generate an error or risk data loss.	OK	An alarm setting must be between 1 and 99.
Confirmation	?	Confirm an action or suggest options.	OK, Cancel	Change settings before switching applications? (For example, when pressing an application key with an open dialog box.)
Warning	!	Ask if user wishes to continue a potentially dangerous action.	OK, Cancel	Are you sure you want to delete this entry?
Error	(stop sign)	Inform user that an attempted action has generated an error and/or cannot be completed.	OK	Disk full.

Button Resource

Name	tBTN
UI Structure	`ControlType`
UI Name	Button
Overview	A button is a clickable UI object is often used to trigger events in an application. A button displays as a text label surrounded by a rectangular frame. The frame has rounded corners. The label may be regular text or a glyph from one of the symbol fonts provided with your development environment (for example, an arrow):



Attributes	Button ID	Developer-defined ID to identify the object. Valid values: 0 – 9999
	Left	Window-relative position of left side of button. Valid values: 0 – 159
	Top	Window-relative position of top of object. Valid values: 0 - 159
	Width	Width of button in pixels. Size the buttons to allow 3–6 pixels of white space at each end of the label. Valid values: 0 – 160
	Height	Height of the button in pixels. Should be 3 pixels larger than the font size (for example, height = 12 for 9-point labels). Valid values: 1 – 160
	Usable	A nonusable object is not considered part of the application interface, and does not draw. Non-usable objects can programmatically be set to usable. Valid values: true (usable), false (nonusable)

Left anchor	Controls how the object resizes itself when its text label is changed. If the attribute is TRUE, the left bound of the object is fixed; if FALSE, the right bound is fixed. Valid values: true (left bound fixed) false (right bound fixed)
Frame	If set to true, a rectangular frame with rounded corners is drawn around the button. Most buttons have frames. Buttons whose labels are single symbol characters, such as scroll buttons, don't have frames. Valid values: true (framed) false (not framed)
Non-bold frame	If set to TRUE, a one-pixel-wide rectangular frame with rounded corners is drawn around the button. If set to false, a bold frame (two pixels wide) is drawn around the button. Nonbold frames are standard. Valid values: true (one-pixel-wide frame) false (two-pixel-wide frame)
Font	ID of font used to draw the text label of the button Valid values: 0 (9-point font) 1 (9-point bold font) 2 (12-point font)
Label	Text displayed inside the button: one line of text, or a single character from a symbol font to create an increment arrow.

Comments

The label is centered inside the button. If the label text is wider than the button, the whole label is centered and both the right and left sides are clipped.

Place command buttons at the bottom of table views and dialog boxes. Leave three pixels between the dialog bottom and buttons.

Increment arrows are a special case; they are buttons that let users increment the value displayed in a data field.

To create an increment arrow, use an arrow character from the symbol font as a label. Several arrow styles and sizes are available.

Event Flow for Button Resource

User Action	System Response
Pen goes down on a button.	`penDownEvent` (store x and y coordinates in `EventType`). `ctlEnterEvent` (store button ID number in `EventType`).
Pen is lifted from button.	`ctlSelectEvent` (store button ID number in `EventType`). `ctlSelectEvent` can be triggered only if a `ctlEnterEvent` with the same button ID number has just occurred. `penUpEvent` (store x and y coordinates in `EventType`).
Pen is lifted outside button.	Nothing happens.

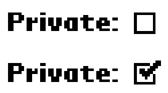
Check Box Resource

Name	tCBX
UI Name	Check box
UI Structure	`ControlType`

Overview

A check box is a small, square UI object that lets users turn something on or off; for example, an alarm. A check box displays a check to indicate it's on, and an optional text label to the right of the box.

The figure below shows a checked and an unchecked check box with a label to the left.



Attributes	Check Box ID	Developer-defined ID to identify the object. Valid values: 0 – 9999
	Left	Window-relative position of left side of object. Valid values: 0 – 159
	Top	Window-relative position of top of object. Valid values: 0 – 159
	Width	Width of the picking area around the check box. Valid values: 0 – 160
	Height	Height of the picking area around the check box. Valid values: 1– 160
	Usable	A nonusable object is not considered part of the application interface, and does not draw. Nonusable objects can programmatically be set to usable. Valid values: true (usable), false (nonusable)
	Selected	Initial selection state of the checkbox. Valid values: true (checked) false (unchecked)
	Group	Group ID of a check box that is part of an exclusive group. Ungrouped (nonexclusive) check boxes have zero as a group ID. Valid values: 0 – 65535
	Font	ID of the font used to draw the text label. Valid values: 0 (9-point font) 1 (9-point bold font) 2 (12-point font)
	Label	Text displayed to the right of the check box. This text is part of the activation area. To create a (nonactive) label on the left of the check box, leave this attribute blank and create a separate Label resource.

Comments	Make sure that only one check box in a group is initially checked. All check boxes are the same size. The bounds determine the toggle area (the screen area the user needs to press to check or uncheck the box). If a label attribute is defined, it's part of the activation area.

Event Flow for Check Box Resource

User Action	System Response
Pen goes down on check box.	`penDownEvent` (store x and y coordinates in `EventType`). `ctlEnterEvent` (store check box's ID number in `EventType`). • If the check box is unchecked, a check appears. • If the check box is already checked, and is grouped, there is no change in appearance. • If the check box is already checked, and is ungrouped, the check disappears.
Pen is lifted from check box.	`ctlSelectEvent` (store check box's ID number in `EventType`, switch check box on (1) or off (0) internally). A `ctlSelectEvent` can be triggered only if a `ctlEnterEvent` with the same check box ID number has just occurred. `penUpEvent` (store x and y coordinates in `EventType`).
Pen is lifted outside button.	Nothing happens.

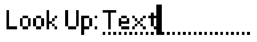
Field Resource

Name	tFLD
UI Name	Field
UI Structure	`FieldType`

Overview

The field UI object is for user data entry in an application. It displays one or more lines of editable text. A field can be underlined, justified left or right, and selectable or unselectable.

The following is an underlined, left-justified field containing data:



Attributes

Field ID	Developer-defined ID to identify the object. Valid values: 0 – 9999
Left	Window-relative position of left side of object. Valid values: 0 – 159
Top	Window-relative position of top of object. Valid values: 0 – 159
Width	Width of the object in pixels. Valid values: 0 – 160
Height	Height of the object in pixels. Valid values: 1– 160
Usable	A nonusable object is not considered part of an application interface and does not draw. Nonusable objects can programmatically be set to usable. Valid values: true (usable), false (nonusable)
Editable	Noneditable fields don't accept user input but can be changed programmatically. Valid values: true (editable), false (noneditable)

Underlined	If set, each line of text is underlined with a gray line.
Single Line	If set to TRUE, the field doesn't scroll horizontally and doesn't accept Return or Tab characters. Only a single line of text is displayed. If the user attempts to enter text beyond this, the system beeps. See Comments for more information on multiline fields. Valid values: true (single line) false (multiline)
Dynamic Size	If TRUE, the height of the field is expanded or compressed as characters are added or removed. Set this attribute to FALSE if the Single Line attribute is set. Valid values: true (dynamically resizes) false (doesn't dynamically resize)
Left Justified	Text justification. Supported only for fields that have the Single Line attribute set to TRUE. Valid values: true (left-justified)—recommended false (right-justified)
Max chars	Maximum number of characters the field accepts. This is a limit on the number of characters a user can enter, but not on what can be displayed. All fields can display up to 32767 characters regardless of this setting. Valid values: 0 – 32767
Font	ID of the font used to draw the text Valid values: 0 (9-point font) 1 (9-point bold font) 2 (12-point font)

Comments

Text fields can be located anywhere but in menus and in the command button area.

Multiline text fields expand. An empty field may display one or more blank lines; for example, records in a To Do list or a text page.

Event Flow for Field Resource

User Action	System Response
Pen goes down on a field.	`penDownEvent` (store x and y coordinates in `EventType`). `fldEnterEvent` (store the field's ID number in EventType).
Pen is lifted	`penUpEvent` (store x and y coordinates in `EventType`). A field remains selected until another field is selected or the form that contains the field is closed.
User enters characters into selected field.	`keyDownEvent` (store ASCII value in `EventType`).

Form Bitmap Resource

Name	tFBM
UI Name	Form bitmap
Overview	Places predefined bitmaps on a given form. Used for icons in Alert dialogs for warnings, errors, information, etc.

Attributes	X Position	Left bounds of bitmap.
	Y Position	Top bounds of bitmap.
	Bitmap Rsc ID	ID of a PICT resource containing the graphic.
	Usable	Set to TRUE if the bitmap should be drawn.

Form Resource

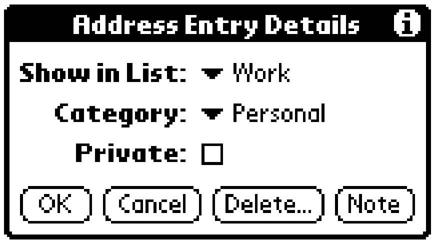
Name	tFRM
UI Name	Form

Overview

A form is a container for one or more of the following UI objects:

- Bitmap
- Button
- Check box
- Field
- Gadget
- Graffiti shift state indicator
- Help string for tips (if modal)
- Label
- List
- Menu bar
- Popup trigger
- Push button
- Repeating button
- Selector trigger
- Table
- Title

An application may contain several different forms the user can trigger from buttons or other control UI objects. A form is a container for other UI objects. Most UI objects are displayed only if they are contained within a form.


Address Entry Details
Show in List: Work
Category: Personal
Private: ☐

Attributes	Left	Window-relative position of left side of object. Valid values: 0 – 159
	Top	Window-relative position of top of object. Valid values: 0 – 159
	Width	Window-relative position of width of the object. Valid values: 0 – 160
	Height	Height of the button in pixels. Valid values: 1– 160
	Usable	A nonusable object is not considered part of the application interface, and does not draw. Nonusable objects can programmatically be set to usable. Valid values: true (usable), false (nonusable)
	Modal	Indicates if the form is modal. Modal forms ignore pen events outside their boundaries.
	Save behind	If this field contains a non-zero value, the region obscured by the form is saved when it is drawn and restored when it is erased. Valid values: zero (don’t save), non-zero (save)
	Form ID	Developer-defined ID used to identify the object. Valid values: 0 – 9999
	Help Rsc ID	Contains the resource ID of a string resource (tSTR) that is the help text from the dialog box. Currently only modal dialogs have a help resource. Valid values: 0 – 9999
	Default Button ID	Defines the ID of the default button. The system simulates pressing the default button where it dismisses the form automatically; for example, when it switches to another application.

Object ID	Resource ID of the form. Valid values: 0 – 9999
Object Type	The resource ID of one or more UI objects. Valid values: tFBM - Bitmap tBTN - Button tCBX - Check box tFLD - Field tGDT - Gadget TGSI - Graffiti shift state indicator tLBL - Label tLST - List tPUT - Popup trigger tPBN - Push button tREP - Repeating button tSLT - Selector trigger tTBL - Table tTTL - Title

Comments

The total display is 160 pixels by 160 pixels. If you want your whole form to be seen, make sure it fits within the display area.

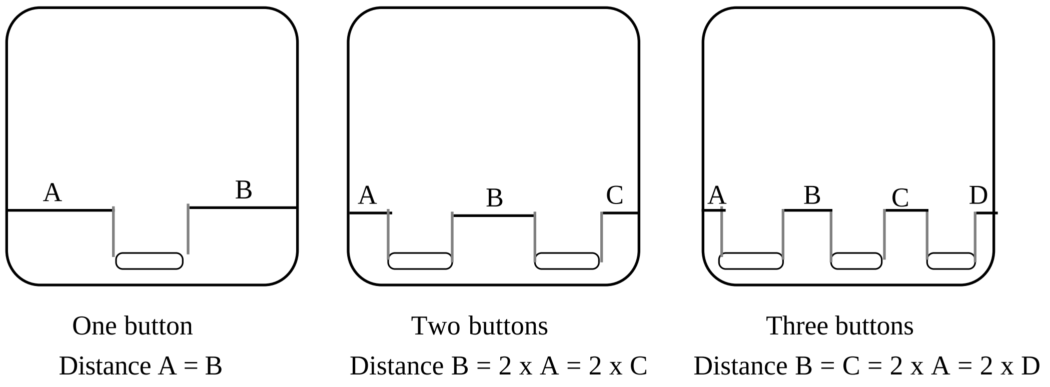
There is an entry in the form resource for each object contained in the form. An entry consists of a resource ID and resource type. The label “1) *****” is a placeholder for the first entry. Select this label and press Command-K to add a choice to the list.

Here are some general design guidelines:

- Each form has a title that displays the name or view of the application (or both.)
- Scroll indicators appear and disappear dynamically, depending on the size of the object. Place them to the right of command buttons.
- Modal dialogs always occupy the full width of the screen and are justified to the bottom of the screen. They hide the command buttons of the base application but don’t obscure the title bar of the base application if possible. There should be a minimum of three pixels between the top of the modal dialog title bar and the bottom of the application title bar. If the dia-

log is too large to accommodate this, the entire application title bar should be obscured.

- Screen command buttons should always be at the bottom of the screen.
- Dialog command buttons appear four pixels above the bottom of the dialog box frame (two-pixel default ring is three pixels above the bottom, and the baseline of the text within the buttons should be aligned).
- Command buttons should be centered so that the spaces between the buttons are twice the width of the spaces between the edges and the border. (See diagram below.) If possible, all buttons should be the same width.



Event Flow

When a form is opened, a `frmOpenEvent` is triggered and the form's ID is stored. A `winEnterEvent` is triggered whenever a form is opened and a `winExitEvent` is triggered whenever a form is closed.

Gadget Resource

Name	tGDT
UI Name	Gadget
Overview	A gadget object lets developers implement a custom UI gadget. The gadget resource contains basic information about the custom gadget which is useful to the gadget writer for drawing and processing user input.

Attributes

Gadget ID	Developer-defined ID used to identify the object.
Left	Window-relative position of left side of object. Valid values: 0 – 159
Top	Window-relative position of top of object. Valid values: 0 – 159
Width	Window-relative position of width of the object. Valid values: 0 – 160
Height	Height of the gadget in pixels. Valid values: 1– 160
Usable	A nonusable object is not considered part of the application interface, and does not draw. Nonusable objects can programmatically be set to usable. Valid values: true (usable), false (nonusable)

Graffiti Shift Resource

Name	tGSI
UI Name	Graffiti® Shift
Overview	Indicates the window- (form) relative position of the Graffiti shift state indicator. The different states are punctuation, symbol, uppercase shift, and uppercase lock. These should appear at the bottom-right of every form that has an editable text field.
Attributes	x pos: Left bounds of text, relative to the form.
	y pos: Top bounds of text, relative to the form.

Label Resource

Name	tLBL
UI Name	Label
Overview	A label is used to display noneditable text or labels on a form (dialog box or full-screen).
Comments	Pressing return in a label wraps the text to the next line.

Attributes

Label ID	Developer-defined ID to identify the object. Valid values: 0 – 9999
Left	Window-relative position of left side of object. Valid values: 0 – 159
Top	Window-relative position of top of object. Valid values: 0 – 159

Usable	A nonusable object is not considered part of an application interface, and does not draw. Nonusable objects can programmatically be set to usable. Valid values: true (usable), false (nonusable)
Font ID	ID of the font used to draw the text. Valid values: 0 (9-point font) 1 (9-point bold font) 2 (12-point font)
Text	Text of the label.

List Resource

Name	tLST
UI Name	List box
UI Structure	`ListType`

Overview

A list box is a UI object you can use to provide a box with a list of choices to the user. The list is scrollable if there are more choices than the size of the list box allows.

A list appears as a vertical list of choices surrounded by a rectangular frame. The current selection of the list is inverted. Arrows for scrolling the list appear in the right margin if necessary.

Lists may also appear as popup lists when used with popup triggers. See Popup List Resource and Popup Trigger Resource.

Attributes

List ID	Developer-defined ID to identify the object. Valid values: 0 – 9999
Left	Window-relative position of left side of object. Valid values: 0 – 159
Top	Window-relative position of top of object. Valid values: 0 – 159

Width	Width of the list. Valid values: 0 – 160
Usable	A nonusable object is not considered part the application interface, and does not draw. Nonusable objects can programmatically be set usable. Valid values: true (usable), false (nonusable)
Font ID	ID of the font used to draw the text. Valid values: 0 (9-point font) 1 (9-point bold font) 2 (12-point font)
Visible items	Height of the list box, in items (choices). For example, if the list has six items but only four fit, specify four.
Items	Items in the list. There is an entry in the resource for each item. The label “1) *****” is the placeholder for the first entry. Select this label and press Command-K to create the first item in the list.

Comments Errors may occur if the number of visible items is greater than the actual number of items. An item’s text is not clipped against the list box’s borders. Set a list box to not usable if it’s linked to a popup trigger.

Use a list to let users choose between items of data; use a menu to activate a command. If a list becomes too tall to fit below the trigger, it is justified up. If it becomes to large for the screen, it scrolls.

Event Flow for List Resource

User Action	System Response
Pen goes down on a list box.	`penDownEvent` (store x and y coordinates in `EventType`). `lstEnterEvent` (store list ID and selected item in `EventType`).
Pen is lifted from the list box.	`lstSelectEvent` is triggered (store button’s ID number and number of selected item in EventType). `penUpEvent` (store x and y coordinates in EventType).

Popup List Resource

Name	tPUL
UI Name	Popup list
Overview	A popup list links a popup trigger to a list box. The popup list itself is not visible.

Attributes

Control ID	Developer-defined ID of a popup trigger object. Valid values: 0 – 9999
List ID	Developer-defined id of a list object. Valid values: 0 – 9999

Comments

To create a popup list and its trigger, you need to also create a Popup Trigger Resource (tPUT) and a List Resource (tLST resource).

Note that this popup resource behaves differently from resources on the Macintosh.

Event Flow

See Event Flow for Popup Trigger Resource.

Popup Trigger Resource

Resource	tPUT
UI name	Popup Trigger
UI structure	`ControlType`

Overview

The popup trigger shows the selection of a list. The user can press the popup trigger to pop up the list and change the selection.

A popup trigger displays a text label and a triangle to the left of the label that indicates the object is a popup trigger.

When the user selects a popup trigger, a list of items pops up.

Attributes	Popup Trigger ID	Developer-defined ID used to identify the button. Valid values: 0 - 9999
	Left	Window-relative position of left side of button. Valid values: 0 – 159
	Top	Window-relative position of top of button. Valid values: 0 – 159
	Width	Width of the button's picking area in pixels. Valid values: 1 – 160
	Height	Height of the button's picking area in pixels. Valid values: 1 –160
	Usable	A nonusable object is not considered part of the application interface, and it does not draw. Nonusable objects can programmatically be set to usable. Valid values: true (usable) false (non-usable)
	Left anchor	Controls how the object resizes itself when its text label is changed. Valid values: true (left bound fixed) false (right bound fixed)
	Font ID	ID of font used to draw text label: Valid values: 0 (9-point font)—Recommended 1 (9-point bold font) 2 (12-point font)
	Label	Text displayed in the popup trigger (right of arrow).

Comments: See also tPUL and tLST.

Event Flow for Popup Trigger Resource

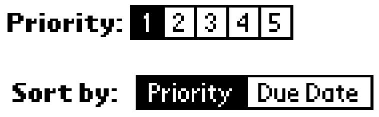
User Action	System Response
Pen goes down on popup trigger.	`penDownEvent` (store x and y coordinates in `EventType`). `ctlEnterEvent` (store popup trigger ID number in `EventType`).
Pen is lifted from popup trigger.	`ctlSelectEvent` (store popup trigger ID number in `EventType`). A `ctlSelectEvent` can be triggered only if a `ctlEnterEvent` with the same popup trigger ID number has just occurred. `winExitEvent` (pass control to a popup list object).
Popup list pops up.	`winEnterEvent` `penUpEvent` (a `penDownEvent` to pop up the popup list).
Pen goes down on item in popup list.	`penDownEvent` occurs.
Pen is lifted from popup list.	`lstSelectEvent` (store the popup list ID and the selected item number in `EventType`). `winExitEvent` causes popup list to disappear; control passes back to the popup trigger. `winEnterEvent` occurs. `popSelectEvent` is triggered if an item was selected in the popup list (store popup trigger ID, the popup list ID, and the item number selected in `EventType`). `penUpEvent` occurs.

Push Button Resource

Name	tPBN
UI Name	Push button
UI Structure	`ControlType`
Overview	Push buttons allow users to select an option from a group of items. The choices should have few characters; if they are long, check boxes are preferable.

Push buttons display a text label surrounded by a 1-pixel-wide rectangular frame. They appear in a horizontal or vertical row with no pixels between the buttons. The buttons share a common border so there appears to be a one pixel line between two controls. The current selection is highlighted.

.



The List By dialog of the Address Book and the Details dialog of the To Do List contain examples of rows of push buttons.

Attributes

Push button ID	Developer-defined ID used to identify the button. Valid values: 0 – 9999
Left	Window-relative position of left side of button. Valid values: 0 – 159
Top	Window-relative position of top of button. Valid values: 0 – 159
Width	Width of the button in pixels. Should be size of label plus two pixels at each end. Valid values: 1 – 160
Height	Height of the button in pixels. Should be font size plus two pixels. Valid values: 1 – 160
Usable	A nonusable object is not considered part of the application interface, and it does not draw. Nonusable objects can programmatically be set to usable. Always mark all buttons in a group usable or nonusable together. Valid values: true (usable) false (nonusable)

Left anchor	Controls how the object resizes itself when its text label is changed. Valid values: true (left bound fixed) false (right bound fixed)
Group	Group ID of a push button that is part of an exclusive group. Only one push button in an exclusive group may be depressed at a time. Ungrouped (non-exclusive) push buttons have zero as a group ID. Valid values: 0 – 65535
Font ID	ID of the font used to draw the text label of the push button. Valid values: 0 (9-point font) —recommended 1 (9-point bold font) 2 (12-point font)
Label	Text displayed inside the push button.

Comment To create a row of push buttons, create a number of individual push button resources with the same height and align them by specifying the same top position for each button.

Event Flow for Push Button Resource

User Action	System Response
Pen goes down on push button.	`penDownEvent` (store x and y coordinates in `EventType`). `ctlEnterEvent` (store push button ID number in `EventType`). Push button is highlighted. If push button is grouped and highlighted, no change. If push button is ungrouped and highlighted, it becomes unhighlighted.
Pen is lifted from push button.	`ctlSelectEvent` (store button ID number and its current state; on = 1; off = 0). `ctSelectEvent` can be triggered only if a `ctlEnterEvent` with the same push button ID number just occurred. `penUpEvent` (store the x and y coordinates).

Repeating Button Resource

Name	tREP
UI Name	Repeat control
Overview	The repeat control object is identical to the button object in its appearance. The repeat control object is used for buttons that need to be triggered continuously by holding the pen down on them.

Attributes

Button ID	Developer-defined ID used to identify the button. Valid values: 0 – 9999
Left	Window-relative position of left side of button. Valid values: 0 – 159
Top	Window-relative position of top of button. Valid values: 0 – 159
Width	Width of the button in pixels. Valid values: 1 – 160
Height	Height of the button in pixels. Valid values: 1 – 160
Usable	A nonusable object is not considered part of the application interface, and it does not draw. Nonusable objects can programmatically be set to usable. Valid values: true (usable) false (nonusable)
Left anchor	Controls how the object resizes itself when its text label is changed. Valid values: true (left bound fixed) false (right bound fixed)
Frame	If set to TRUE, a rectangular frame with rounded corners is drawn around the button. Valid values: true (framed) false (not framed)

Nonbold frame	Determines the width of the rectangular frame drawn around the object. Valid values: true (1-pixel-wide frame) false (2-pixel-wide frame)
Font ID	ID of the font used to draw the text label. Use a single character from one of the Symbol fonts to create increment arrows. Valid values: 0 (9-point font) 1 (9-point bold font) 2 (12-point font)
Label	Text displayed inside the button.

Comments The attributes match those of the Button Resource (tBTN); the behavior differs.

You can also use repeating buttons to create increment arrows. See Button Resource for more information.

Event Flow for Repeating Button Resource

A repeating button is similar in appearance to a button but it generates different events. A button generates a `ctlEnterEvent` when it is pressed and a `ctlSelect` event when it is released. A repeating button generates a `ctlEnterEvent` when it is pressed and a `ctlRepeatEvent` as long as it remains pressed. Here's a more detailed discussion of the events:

User Action	System Response
Pen goes down on a repeating button.	`penDownEvent` (store x and y coordinates in `EventType`). `ctlEnterEvent` (store button's ID number in `EventType`).
Pen remains on repeating button.	For every given amount of time the pen is down on the repeat control object, a `ctlRepeatEvent` is generated.
Pen is dragged off the repeating button.	No additional `ctlRepeateEvent` occurs.

User Action	System Response
Pen is dragged back onto the button.	`ctlRepeatEvent` begins to occur again.
Pen is lifted.	`penUpEvent` (store x and y coordinates in `EventType`)

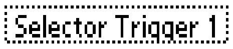
Selector Trigger Resource

Name	tSLT
UI Name	Selector Trigger
UI Structure	`ControlType`

Overview

Users can tap a selector trigger to pop up a dialog that lets them select an item. The selected item becomes the label of the selector trigger. For example, a selector trigger for time pops up a time selector. The selected time is entered into the selector trigger.

A selector trigger displays a text label surrounded by a gray rectangular frame, as follows:



Attributes

Selector Trigger ID	Developer-defined ID used to identify the object. Valid values: 0 – 9999
Left	Window-relative position of the left side of the object. Valid values: 0 – 159
Top	Window-relative position of top of object. Valid values: 0 – 159
Width	Width of the object in pixels. Valid values: 1– 160

Height	Height of the object in pixels. Height extends two pixels above and one pixel below the 9-point plain font. Height is one pixel above command buttons to accommodate the gray frame. Valid values: 1– 160
Usable	A nonusable object is not considered part of the application interface, and it does not draw. Nonusable objects can programmatically be set to usable. Valid values: true (usable) false (nonusable)
Left anchor	Controls how the object resizes itself when its text label is changed. If TRUE, the left bound of the object is fixed, if FALSE the right bound is fixed. Valid values: true (left bound fixed) false (right bound fixed.)
Font ID	ID of the font used to draw the text label. Valid values: 0 (9-point font)—Recommended 1 (9-point bold font) 2 (12-point font)

Event Flow for Selector Trigger Resource

User Action	**System Response**
Pen goes down on a selector trigger.	`penDownEvent` (store x and y coordinates in `EventType`). `ctlEnterEvent` (store selector trigger ID number in `EventType`).
Pen is lifted from the selector trigger.	`ctlSelectEvent` (store selector trigger ID number in `EventType`). A `ctlSelectEvent` can only be triggered if a `ctlEnterEvent` with the same selector trigger ID number has just occurred. `frmOpenEvent` followed by a `winExitEvent`, control is passed to a form object. When control is passed back to the selector trigger, a `winEnterEvent` and a `penUpEvent` occur.

String Resource

Name	tSTR
UI Name	String
Overview	Stores data strings used by the program. String resources may be entered as text strings or as a series of hexadecimal characters.

Attributes

String	The text string to be stored (in decimal ASCII).
Data	The text string to be stored (in hexadecimal ASCII).

Comments The string resource uses either the string or data. If both are entered, they are concatenated.

Table Resource

Name	tTBL
UI Name	Table
Overview	The table object allows the developer to organize a collection of objects on the display. For example, a table may contain a column of labels that correspond to a column of fields.

Attributes

Table ID	Developer defined ID used to identify the object. Valid values: 0 – 9999
Left	Window-relative position of the left side of the object. Valid values: 0 – 159
Top	Window-relative position of top of object. Valid values: 0 – 159
Width	Width of the object in pixels. Valid values: 1– 160

Height	Height of the object in pixels. Valid values: 1–160
Rows	Number of rows in the table.
Columns	Number of columns in the table.
Column width	Width of the nth column.

Comments Since tables are scrollable, they may be larger than the display.

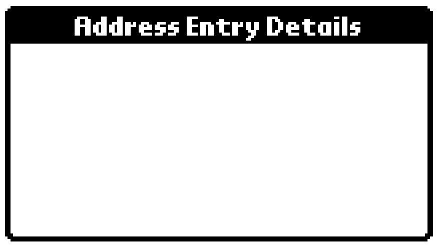
Title Resource

Name tTTL

UI Name Title

Overview This resource lets you place a title at the top of a form (dialog box or view). If the form is modal, the title is drawn centered at the top of the form, if nonmodal the title is drawn left-aligned. A title is drawn within the bounds of the form, not above the form's bounds.

The figure below shows a form with a title.



Attributes

Title	Title string displayed.

Comments The title must be one line; it uses about 13 pixels of the top of the form.

Version Number String

Name	tver
Overview	Specifies the version number display by the “About Dialog Box.”
Attributes	Version string. This consists of a major version number and a minor version number separated by a decimal point.

Supported Fonts

The following fonts are supported:

FontID	Name
0	stdFont
1	boldFont
2	largeFont
3	symbolFont
4	checkboxFont

4

Palm OS Events

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 does not respond to this event by exiting, the system can't start the other application.

ctlEnterEvent

The control routine `CtlHandleEvent` sends this event when it receives a `penDownEvent` within the bounds of a control. The following data is passed with the event:

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 `cltExitEvent` is added to the event queue.

The following data is passed with the event:

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 Repeat control (tREP), it sends a `ctlRepeatEvent`. When `CtlHandleEvent` receives a `ctlRepeatEvent` in a repeat control, it sends another `ctlRepeatEvent` if the pen remains down within the bounds of the control for 1/2 second beyond the last `ctlRepeatEvent`.

The following data is passed with the event:

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.

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 `cltSelectEvent` is added to the event queue; if not, a `ctlExitEvent` is added to the event queue.

The following data is passed with the event:

controlID	Developer-defined ID of the control.
pControl	Pointer to a control structure (`ControlType`).
on	TRUE when the control is depressed; otherwise FALSE.

daySelectEvent

The `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.

The following data is passed with the event:

pSelector	Pointer to a day selector structure (`DaySelectorType`).
selection	Not used.
useThisDate	Set to TRUE to automatically use the selected date.

fldChangedEvent

The field routine `FldHandleEvent` sends this event when the text of a field has been scrolled as a result of drag-selecting. 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.

The following data is passed with the event:

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. The following data is passed with the event:

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

fldHeightChangedEvent

The field routine `FldHandleEvent` sends this event. The field API supports a feature that allows a field to dynamically resize its visible height as text is added or removed from it. Functions in the field API send a `fldHeightChangedEvent` to change the height of a field. Applications don't usually send or handle this event.

The following data is passed with the event:

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 does not intercept this event, the routine `FrmHandleEvent` erases the specified form and releases any memory allocated for it.

The following data is passed with the event:

formID	Developer-defined ID of the form.

frmLoadEvent

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

The application is responsible for handling this event.

The following data is passed with the event:

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.

The following data is passed with the event:

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.

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 and therefore must ask applications to redraw the regions themselves.

An application can also use this event to update a form, usually when changes made to one form need to be reflected in another form.

The following data is passed with the event:

formID	Developer-defined ID of the form.
updateCode	The reason for the update request. `FrmEraseForm` sets this code to zero. Application developers can define their own `updateCode`. The `updateCode` is passed as a parameter to `FrmUpdateForm`.

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).

The following data is passed with the event:

chr	ASCII code of character, or zero if the key is a virtual key code (for example, the Find key).
keyCode	Virtual key code; for example, the Find key.
modifiers	One of the following:

shiftKeyMask	True if Graffiti is in case-shift mode.
capsLockMask	True if Graffiti is in cap-shift mode.
numLockMask	True if Graffiti is in numeric-shift mode.
commandKeyMask	True if the Graffiti glyph was the menu command glyph.
optionKeyMask	Not implemented. Reserved.
controlKeyMask	Not implemented. Reserved.
autoRepeatKeyMask	True if generated due to auto-repeat.
doubleTapKeyMask	Not implemented. Reserved.
poweredOnKeyMask	True if the key press caused the system to be powered on.

lstEnterEvent

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

The following data is passed with the event:

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

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.

The following data is passed with the event:

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.

The following data is passed with the event:

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

menuEvent

The menu routine `MenuHandleEvent` sends this event:

- when the user selects an item from a pull-down menu or
- when the user writes the menu the Graffiti Command shortcut followed by an available command, for example, Command-C for copy

The following data is passed with the event:

itemID	Item ID of the selected menu command, which is the Menu ID assigned to the MENU resource in ResEdit, plus the position of the command in the pull-down menu. For example, for an Edit menu with the Menu ID 300 that contains the commands Undo, Cut, Copy, and Paste, the Item ID of the Copy command is 302.

nilEvent

The event manager sends this event when there are no events in the event queue. This happens only if the routine `EvtGetEvent` is passed a time out value (a value other then `evtWaitForever`, -1). If `EvtGetEvent` is unable to return an event in the specified time, it returns a `nilEvent`. A `nilEvent` is useful for animation, polling, and similar situations.

penDownEvent

The event manager sends this event when the pen first touches the digitizer.

The following data is passed with the event:

penDown	Always true.
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:

penDown	Always true.
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).

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.

The following data is passed with the event:

penDown	Always false.
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).
start	Display-relative start point of the stroke.
end	Display-relative end point of the stroke.

popSelectEvent

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

The following data is passed with the event:

controlID	Developer-defined ID of the resource.
pControl	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.

tblEnterEvent

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

The following data is passed with the event:

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 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.

The following data is passed with the event:

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.

The following data is passed with the event:

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.

The following data is passed with the event:

enterWindow	Pointer to the window we are entering. If the window is a form, this is a pointer to a `FormType` structure; if not, it is a pointer to a `WindowType` structure.
exitWindow	Pointer 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, this is a pointer to a `FormType` structure; if not, it is 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`).

The following data is passed with the event:

enterWindow	Pointer to the window we are entering. If the window is a form then this is a pointer to a `FormType` structure; if not, it is a pointer to a `WindowType` structure.
exitWindow	Pointer to the window we are exiting. If the window is a form then this is a pointer to a `FormType` structure; if not, it is a pointer to a `WindowType` structure.

5

Palm OS UI Objects

A Palm OS UI object is a C structure that is linked with one or more items on the screen. By changing field values of the C structure, an application can manipulate its user interface. Note that Palm UI objects are just structures, not the more elaborate objects found in some systems. This is useful because a C structure is more compact than other objects could be.

This chapter helps you develop your application's user interface by providing information about each object's structure, associated events, associated UI resource files, and all API calls available for manipulating the structure. It discusses these topics:

- Control Objects
- Field Objects
- Form Objects
- List Object
- Menu Objects
- Date and Time UI Objects
- Insertion Point Object
- Table Objects
- Window Objects

Control Objects

This section provides the following information about control objects:

- Control Object Overview
- Control Object Events
- Structure of a Control
- Associated Resources
- Control Functions

Control Object Overview

Control objects are objects you can add to your forms to allow for user interaction. There are six types of control objects:

- **Buttons** display a text label in a box. The default style for a button is a text string centered within a rounded rectangle. Touching a button with the pen highlights the button until the pen is released or dragged outside the bounds of the button. Buttons have rounded corners unless a rectangular frame is specified. A button without a frame inverts a rounded rectangular region when pressed.
- **A popup trigger** displays a text label followed by a graphic element (always on the right) that signifies the control initiates a popup list. If the text label changes, the width of the control expands or contracts to the width of the new label plus the graphic element.
- **A selector trigger** displays a text label surrounded by a gray rectangular frame. If the text label changes, the width of the control expands or contracts to the width of the new label.
- **A repeat control** looks like a button. In contrast to buttons, however, users can repeatedly select repeat controls if they don’t lift the pen when the control has been selected. The object is selected repeatedly until the pen is lifted.
- **Push buttons** look like buttons but the frame always has square corners. Touching a push button with the pen inverts the bounds. If the pen is released within the bounds, the button remains inverted.
- **Check boxes** display a setting, either on (checked) or off (unchecked). Touching a check box with the pen toggles the setting. The check box appears as a square, which contains a check mark if the check box’s setting is on. A check box can have a text label attached to it; selecting the label also toggles the check box.

Push buttons and check boxes can be arranged into exclusive groups; one and only one control in a group can be on at a time.

Control Object Events

Control objects generate four types of events: `ctlEnterEvent`, `ctlExitEvent`, `ctlRepeatEvent`, and `ctlSelectEvent`. All these events are generated by the control event handler (`CtlHandleEvent`).

When `CtlHandleEvent` receives a `penDownEvent` with the pen position in the bounds of the control object, it adds a `ctlEnterEvent` to the event queue.

When `CtlHandleEvent` receives a `ctlEnterEvent`, it inverts the control and tracks the pen until the pen comes up or until the pen is dragged outside the bounds of the control.

- If the pen comes up in the bounds of the control, a `ctlSelectEvent` is added to the event queue.
- If the pen is dragged outside the bounds of the control, the control reverts to its original visual state and a `ctlExitEvent` is added to the event queue.

When `CtlHandleEvent` receives a `ctlEnterEvent`, for a repeat control, it sends a `ctlRepeatEvent`. When a repeat control receives a `ctlRepeatEvent`, it tracks the pen for a period of time and then sends another `ctlRepeatEvent` if the pen is still within the bounds of the control.

When `CtlHandleEvent` receives a `ctlExitEvent`, it tracks the pen until the pen comes up or is dragged inside the bounds of the control. If the pen is dragged into the control, a `ctlEnterEvent` is added to the event queue. If the pen is released outside the control, no event is posted.

All events posted by the control handler contain the ID of the control and a pointer to the control data structure.

Structure of a Control

```
typedef struct {
word              id;
CharPtr           text;
RectangleType     bounds;
ControlAttrType   attr;
ControlStyleType  style;
FontID            font;
byte              group;
} ControlType;

typedef ControlType* ControlPtr;

typedef struct {
Byte usable     :1;
Byte enabled    :1;
Byte visible    :1;
Byte on         :1;
Byte leftAnchor :1;
Byte frame      :3;
} ControlAttrType;

enum controlStyles {buttonCtl, pushButtonCtl,
          checkboxCtl, popupTriggerCtl,
          selectorTriggerClt, repeatingButtonCtl};
typedef enum controlStyles ControlStyleType;

enum buttonFrames {noButtonFrame,
            standardButtonFrame, boldButtonFrame,
            rectangleButtonFrame};

typedef enum buttonFrames ButtonFrameType;
```

Fields of a ControlType Structure

The `id` field holds the symbolic ID of the control that was specified by the application developer. By convention, these should match the Macintosh resource IDs, but this is not mandatory.

The `bounds` field contains the 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.

The `text` field is a pointer to the control's label; if it's NULL the control has no label. Only buttons, push buttons, and text boxes have text labels.

The `attr` field contains the control's attributes. The `attr` field is a bit field that contains the following members: `usable`, `enabled`, `visible`, `on`, `leftAnchor`, and `frame`.

- A control that does not have the `usable` attribute set is not considered to be part of the interface of the current application, and it does not appear on screen.
- A control that does not have the `enable` attribute set appears "grayed out," and does not respond to the pen. This is strongly discouraged because it's a poor use of screen real estate.
- The `visible` attribute is set and cleared internally when the control is drawn and erased.
- The `leftAnchor` attribute is used by controls that expand and shrink their width when their label is changed. If the attribute is set, the left bound of the control is fixed.
- The `frame` field specifies the type of frame drawn around the button controls. Only button controls use this attribute; for all other types of controls the `ControlStyle` determines the frame.

The `style` field holds the control's style: button, push button, check box, popup trigger, popup selector, or repeating button. (See the `ControlStyleType` enum listed under Structure of a Control.)

The `font` field specifies the font to use to draw the control's label.

The `group` field contains the group ID of a push button or a check box that is part of an exclusive group. The control routines do not automatically turn one control off when another is hit. It is up to the application or a higher-level object, like a dialog box, to manage this.

Associated 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)

Control Functions

- `CtlDrawControl`
- `CtlEraseControl`
- `CtlGetLabel`
- `CtlGetValue`
- `CtlHandleEvent`
- `CtlHideControl`
- `CtlHitControl`
- `CtlEnabled`
- `CtlSetEnabled`
- `CtlSetLabel`
- `CtlSetUsable`
- `CtlSetValue`
- `CtlShowControl`

Field Objects

This section provides the following information about field objects:

- Field Object Overview
- Field Object Events
- Structure of a Field
- Associated Resources
- Field Functions

Field Object Overview

A field object displays one or more lines of editable text, supporting these features:

- Proportional fonts (only one font per field)
- Drag-selection
- Scrolling for multiline fields
- Cut, copy, and paste
- Left and right text justification
- Tab stops
- Editable/noneditable attribute
- Expandable field height (the height of the field expands as more text is entered)
- Underlined text (each line of the field is underlined.)
- Maximum character limit (the field stops accepting characters when the maximum is reached)
- Special keys (Graffiti strokes) to support cut, copy, and paste
- Insertion point positioning with pen (the insertion point is positioned by touching the pen between characters)

The field object does **not** support overstrike input mode; scroll bars; horizontal scrolling for single line fields; word selection; character filters (for example, only numeric characters accepted); numeric formatting; or special keys for page up, page down, left word, right word, home, end, left margin, right margin, and backspace.

Field Object Events

Events in field objects are handled by `FldHandleEvent`. `FldHandleEvent` handles events of type `penDownEvent`, `fldEnterEvent`, and `keyDownEvent`.

When `FldHandleEvent` receives a `penDownEvent` with the pen position within the bounds of the field, it adds a `fldEnterEvent` to the event queue.

When `FldHandleEvent` receives a `fldEnterEvent`, it sets the insertion point position to the position of the pen and tracks the pen until it is released. Drag-selection and drag-scrolling are supported.

A `keyDownEvent` with the following special characters is handled by `FldHandleEvent`:

- up arrow—Move insertion point up a line.
- down arrow—Move insertion point down a line; the insertion point doesn't move beyond the last line that contains text.
- left arrow—Move insertion point one character position to the left; when the left margin is reached move to the end of the previous line.
- right arrow—Move insertion point one character position to the right; when the right margin is reached move to the start of the next line.
- cut key—Cut the current selection to the text clipboard.
- copy key—Copy the current selection to the text clipboard.
- paste key—Insert clipboard text into the field at insertion point.

Structure of a Field

```
typedef struct {
Word              id;
RectangleType     rect;
FieldAttrType     attr;
CharPtr           text;
VoidHand          textHandle;
LineInfoPtr       lines;
Word              textLen;
Word              textBlockSize;
Word              maxChars;
Word              selFirstPos;
Word              selLastPos;
Word              insPtXPos;
Word              insPtYPos;
FontID            fontID;
} FieldType;

typedef FieldType* FieldPtr;
typedef struct {
Word usable       :1;
Word visible      :1;
Word editable     :1;
Word singleLine   :1;
Word hasFocus     :1;
Word dynamicSize  :1;
Word insPtVisible :1;
Word dirty        :1;
Word underlined   :2;
Word justification:2;
} FieldAttrType;

typedef struct {
Word        start;
Word        length;
} LineInfoType;
```

```
typedef LineInfoType* LineInfoPtr;
```

Fields of a Field Structure

The `id` field holds an ID value specified by the application developer. This ID value is included as part of the event data of `fldEnterEvent`.

The `rect` field contains the position and size of the field object.

The `attr` field contains the field object's attributes. The `attr` field is a bits field that contains the following members: `usable`, `visible`, `editable`, `singleLine`, `hasFocus`, `dynamicSize`, `insPtVisible`, `dirty`, `underlined`, and `justification`.

- A field object that does not have the `usable` attribute set is not considered part of the current interface of the application, and it doesn't appear on screen.
- The `visible` attribute is set or cleared internally when the field object is drawn or erased.
- A field object that does not have its `editable` attribute set does not accept Graffiti input or edit commands and the insertion point cannot be positioned with the pen.
- If the `singleLine` attribute is set, the height of the `singleLine` field doesn't expand to accommodate more text.
- The `hasFocus` attribute is set internally when the field has the current focus. The blinking insertion point appears in the field that has the current focus.
- If the `dynamicSize` attribute is set, the height of the field expands as characters are entered into the field.
- If the `insPtVisible` attribute is set, the insertion point is scrolled into view. This attribute is set and cleared internally.
- If a field has its `dirty` attribute set, the user has modified the field.
- If a field has its `underlined` attribute set each line of the field, including blank lines, is underlined.
- The `justification` attribute specifies the text alignment (left or right justification only; center justification is not supported).

The `text` field holds a pointer to the null-terminated string that is displayed by the field object.

The `textHandle` field contains the handle to the stored text.

The `lines` field holds a pointer to an array of `LineInfoType` structures. There is one entry in this array for each visible line of the text. The `LineInfoType` structure contains the character position, in the field's text string, of the first character displayed by a line and the number of characters displayed.

The `textLen` field holds the current number of characters in the string displayed by the field object, the null-terminator is excluded.

The `textBlockSize` field holds the allocated size of the memory block that holds the field object's text string.

The `maxChars` field specifies the maximum number of characters the field object accepts.

The `selFirstPos` field is the starting character position of the current selection.

The `selLastPos` field is the ending character position of the current selection. When `selFirstPos` equals `selLastPos,` there is no selection.

The `insPtXPos` contains the column position of the insertion point.

The `insPtYPos` contains the display line where the insertion point is positioned.

The first display line is zero.

Associated Resources

The Field Resource (tFLD) represents a field on screen.

Field Functions

- `FldCalcFieldHeight`
- `FldCompactText`
- `FldCopy`
- `FldCut`
- `FldDelete`
- `FldDirty`
- `FldDrawField`
- `FldEraseField`
- `FldFreeMemory`
- `FldGetAttributes`
- `FldGetBounds`
- `FldGetFont`
- `FldGetInsPtPosition`
- `FldGetMaxChars`
- `FldGetScrollPosition`
- `FldGetSelection`
- `FldGetTextAllocatedSize`
- `FldGetTextHandle`
- `FldGetTextHeight`
- `FldGetTextLength`
- `FldGetTextPtr`
- `FldGetVisibleLines`
- `FldGrabFocus`
- `FldHandleEvent`
- `FldInsert`
- `FldMakeFullyVisible`
- `FldPaste`
- `FldRecalculateField`
- `FldReleaseFocus`
- `FldScrollable`
- `FldScrollField`
- `FldSendChangeNotification`

- `FldSendHeightChangeNotification`
- `FldSetAttributes`
- `FldSetBounds`
- `FldSetDirty`
- `FldSetFont`
- `FldSetFont`
- `FldSetInsPtPosition`
- `FldSetMaxChars`
- `FldSetScrollPosition`
- `FldSetSelection`
- `FldSetText`
- `FldSetTextAllocatedSize`
- `FldSetTextHandle`
- `FldSetTextPtr`
- `FldSetUsable`
- `FldUndo`
- `FldWordWrap`

Form Objects

This section provides the following information about form objects:

- Form Object Overview
- Structure of a Form
- Associated Resources
- Form Functions

Form Object Overview

A form object is used as a container for all other UI objects. A form is a window and everything contained within it. Events in form objects are handled by the `FrmHandleEvent` routine.

When `FldHandleEvent` receives a `penDownEvent` with the pen position within the bounds of the form object, it checks the list of objects contained by the form to determine if the pen is within the bounds of one of the objects. If it is, the appropriate handler is called to handle the event, for example, if the pen is in a control, `CtlHandleEvent` is called.

When `FrmHandleEvent` receives a `keyDownEvent`, it passes the event to the handler for the object that has the focus. If no object has the focus, the event is ignored.

When `FldHandleEvent` receives a `ctlEnterEvent`, it checks if the control is in an exclusive control group. If it is, it deselects the currently selected control of the group and passes the event to a pointer to the object the event occured in to `CtlHandleEvent`. The object pointer is obtained from the event data.

When `FldHandleEvent` receives a `ctlRepeatEvent`, it passes the event and a pointer to the object the event occured in to the appropriate handler. The object pointer is obtained from the event data.

When `FldHandleEvent` receives a `ctlSelectEvent`, it checks if the control is a popupTriggerCtl. If it is, the list associate 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 `popSelectEvent` is added to the event queue.

When `FldHandleEvent` receives a `popSelectEvent`, it sets the label of the popup trigger to the current selection of the popup list.

When `FldHandleEvent` receives a `lstEnterEvent` or `tblEnterEvent`, it passes the event and a pointer to the object the event occurred in to the appropriate handler. The object pointer is obtained from the event data.

When `FldHandleEvent` receives a `fldEnterEvent` or `fldHeightChangedEvent`, it checks if a field object or a table object has the focus and passes the event to the appropriate handler. The table object is also a container object, which may contain a field object. If `TblHandleEvent` receives a field event, it passes the event to the field object contained within it.

When `FldHandleEvent` receives a `frmCloseEvent`, it erases the form and releases any memory allocated for it.

When `FldHandleEvent` receives a `frmUpdateEvent`, it redraws the form.

When `FldHandleEvent` receives a `menuEvent`, it checks if the menu command is one of the system edit menu commands. The system provides a standard edit menu which contains the commands Undo, Cut, Copy, Paste, Select All, and Keyboard. `FldHandleEvent` responds to these commands.

Structure of a Form

```
typedef struct {
WindowType          window;
Word                formId;
FormAttrType        attr;
WinHandle           bitsBehindForm;
FrmEventHandlerPtr  handler;
Word                focus;
Word                defaultButton;
Word                helpRscId;
Word                menuRscId;
Word                numObjects;
FormObjListType*    objects;
} FormType;

typedef FormType * FormPtr;

typedef struct {
Word usable        :1;
Word enabled       :1;
Word visible       :1;
Word dirty         :1;
Word saveBehind    :1;
Word graffitiShift :1;
Word reserved      :11;
} FormAttrType;

typedef struct {
FormObjectKind  objectType;
FormObjectType  object;
} FormObjListType;
```

```
typedef union {
void *                   ptr;
FieldType*               field;
ControlType*             control;
ListType*                list;
TableType*               table;
FormBitmapType*          bitmap;
FormLabelType *          label;
FormTitleType*           title;
FormPopupType*           popup;
FormGraffitiStateType*   grfState;
FormGadgetType*          gadget;
} FormObjectType;

enum formObjects {
frmFieldObj,
frmControlObj,
frmListObj,
frmTableObj,
frmBitmapObj,
frmLineObj,
frmFrameObj,
frmRectangleObj,
frmLabelObj,
frmTitleObj,
frmPopupObj,
frmGraffitiStateObj,
FrmGadgetObj};

typedef enum formObjects FormObjectKind;

typedef struct {
Word usable :1;       // Set if part of ui
} FormObjAttrType;
```

```
typedef struct {
FormObjAttrType attr;
PointType       pos;
Word            rscID;
} FormBitmapType;

typedef struct {
FormObjAttrType   attr;
PointType         point1;
PointType         point2;
} FormLineType;

typedef struct {
Word            id;
FormObjAttrType attr;
RectangleType   rect;
Word            frameType;
} FormFrameType;

typedef struct {
FormObjAttrType attr;
RectangleType   rect;
} FormRectangleType;

typedef struct {
Word            id;
PointType       pos;
FormObjAttrType attr;
FontID          fontID;
char *          text;
} FormLabelType;

typedef struct {
RectangleType rect;
char *        text;
} FormTitleType;
```

```
typedef struct {
unsigned short  controlID;
unsigned short  listID;
} FormPopupType;

typedef struct{
PointerType   pos;
}FrmGraffitiStateType;

typedef struct{
Word            id;
FormObjAttrType attr;
RectangleType   rect;
VoidPtr         date;
}FormGadgetType;
```

Fields of Form Objects

The `window` field is the structure of the window object that corresponds to the form.

The `formId` field contains the 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 Macintosh resource ID.

The `attr` field contains form object's attributes. The `attr` field is a bit field that contains the members: `usable`, `enable`, `visible`, `dirty`, `saveBehind`, and `reserved`.

- If `usable` is set TRUE, the form is considered part of the user interface.
- The `enable` attribute specifies whether or not the user can interact with the form.
- If `visible` is set, the form is drawn.
- The `dirty` attribute is set if the form has been modified in any way. Modifications include the changing of a field or check box (not currently supported).
- The `saveBehind` attribute is set if the bits behind the form are to be saved when the form is drawn.

The `bitsBehindForm` field is used to save all the bits behind the form so the screen can be properly refreshed when the form is closed.

The `focus` field contains the 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.

The `handler` field contains the routine called when the form needs to handle an event, typically set by the application in the `ApplicationHandleEvent` function.

The `defaultButton` field contains the index of the object defined as the default button. This value is used by the routine `FrmDoDialog`

The `helpRscId` field contains the resource ID number of the help resource. The help resource is of type tSTR.

The `MenuRscId` field contains the ID number of a menu bar to use if the form is a menu, or zero if the form is not a menu.

The `numObjects` field specifies the number of objects contained within the form.

The `objects` field contains a pointer to the array of objects contained within the form.

Associated Resource

The Form Resource (tFRM) is used to represent forms on screen.

Form Functions

- `FrmAlert`
- `FrmCloseAllForms`
- `FrmCopyLabel`
- `FrmCopyTitle`
- `FrmCustomAlert`
- `FrmDeleteForm`
- `FrmDispatchEvent`
- `FrmDoDialog`
- `FrmDrawForm`

- `FrmEraseForm`
- `FrmGetActiveForm`
- `FrmGetActiveFormID`
- `FrmGetControlGroupSelection`
- `FrmGetControlValue`
- `FrmGetFirstForm`
- `FrmGetFocus`
- `FrmGetFormBounds`
- `FrmGetFormId`
- `FrmGetFormPtr`
- `FrmGetGadgetData`
- `FrmGetLabel`
- `FrmGetNumberOfObjects`
- `FrmGetObjectBounds`
- `FrmGetObjectId`
- `FrmGetObjectIndex`
- `FrmGetObjectPositon`
- `FrmGetObjectPtr`
- `FrmGetObjectType`
- `FrmGetTitle`
- `FrmGetUserModifiedState`
- `FrmGetWindowHandle`
- `FrmGotoForm`
- `FrmHandleEvent`
- `FrmHelp`
- `FrmHideObject`
- `FrmInitForm`
- `FrmPopupForm`
- `FrmReturnToForm`
- `FrmSaveAllForms`
- `FrmSetActiveForm`
- `FrmSetCategoryLabel`
- `FrmSetControlGroupSelection`

- `FrmHideObject`
- `FrmInitForm`
- `FrmPopupForm`
- `FrmReturnToForm`
- `FrmSaveAllForms`
- `FrmSetActiveForm`
- `FrmSetCategoryLabel`
- `FrmSetControlGroupSelection`
- `FrmSetControlValue`
- `FrmSetEventHandler`
- `FrmSetFocus`
- `FrmSetGadgetData`
- `FrmSetNotUserModified`
- `FrmSetObjectPositon`
- `FrmSetTitle`
- `FrmShowObject`
- `FrmUpdateScrollers`
- `FrmUpdateForm`
- `FrmVisible`

List Object

This section provides information about list objects by discussing these topics:

- List Object Overview
- List Object Events
- Structure of a List
- Associated Resources
- List Functions

List Object Overview

The list object appears as a vertical list of choices in a box. The current selection of the list is inverted. If there are more choices than can be displayed, the system draws small arrows (scroll indicators) in the right margin next to the first and last visible choice.

When the pen comes down and up on a scroll indicator, the list is scrolled. When the user scrolls down, the last visible item becomes the first visible item, if there are enough items to fill the list. If not, the list is scrolled so that the last item of the list appears at the bottom of the list. The reverse is true for scrolling up. Scrolling does not change the current selection.

Bringing the pen down on a list item unhighlights the current selection and highlights the item under the pen. Dragging the pen through the list highlights the item under the pen. Dragging the pen above or below the list causes the list to scroll if it contains more choices than are visible.

When the pen is released over an item, that item becomes the current selection. When the pen is dragged outside the list, the item that was highlighted before the `penDownEvent` is highlighted again if it's visible. If it's not, no item is highlighted.

List Object Events

The list object generates two types of event structures: `lstEnterEvent` and `lstSelectEvent`. Both events are generated by the list event-handler function `LstHandleEvent`.

When `LstHandleEvent` receives a `penDownEvent`, it adds a `lstEnterEvent` to the event queue if the pen position is within the bounds of the list.

When `LstHandleEvent` receives a `lstEnterEvent`, it tracks the pen until it's released. If the pen is released on a list choice, a new selection is made (the data structure is modified) and a `lstSelectEvent` is added to the event queue. If the pen is released outside the list, the selection is unchanged and no event is posted.

A `lstEnterEvent` contains the following data:

- list ID
- a pointer to the list object

- the item the pen is on

A `lstSelectEvent` contains the following data:

- list ID
- a pointer to the list object
- the item selected

Structure of a List

```
typedef struct {
Word                 id;
RectangleType        bounds;
ListAttrType         attr;
CharPtr*             itemsText;
Word                 numItems;
Word                 currentItem;
Word                 topItem;
FontID               font;
WinHandle            popupWin;
ListDrawDataFuncPtr  drawItemCallback;
} ListType;

typedef struct {
unsigned usable   :1;
unsigned enabled  :1;
unsigned visible  :1;
unsigned poppedUp :1;
unsigned reserved :4;
} ListAttrType;
```

List Object Fields

The `id` field holds an ID value, specified by the application developer. This ID value is part of the event data of `lstEnterEvent` and `lstSelectEvent`.

The `bounds` field contains the bounds of the list, relative to the window.

The `attr` field contains the list's attributes. The `attr` field is a bit field that contains the following members: `usable`, `enable`, `visible`, and `poppedUp`:

- A list that does not have the `usable` attribute set is not considered part of the current interface of the application, and it doesn't appear on screen.
- The `enable` attribute is set if users can interact with the list.
- The `visible` attribute is set or cleared internally when the list is drawn or erased.
- The `poppedUp` attribute is set if the choices are displayed in a popup window. This attribute is set and cleared internally.

The `itemsText` field holds a pointer to an array of pointers to the text of the choices.

The `font` field holds the ID of the font used to draw all list text strings.

The `popupWin` is the handle of the window created when a list is displayed if the `poppedUp` attribute is set.

The `drawItemsCallback` is the function used to draw an item in the list. If null, the default drawing routine is used instead.

```
void ListDrawDataFuncType
              (UInt itemNum,
              RectanglePtr bounds,
              CharPtr *itemsText)
```

The `numItems` field contains the number of choices in the list.

The `currentItem` field holds the currently selected list choice (0 = first choice).

The `topItem` field holds the first choice displayed in the list.

Associated Resources

The resources tLST (List Resource), tPUL (Popup List Resource), and tPUT (Popup Trigger Resource) are used together to represent an active list.

List Functions

- `LstDrawList`
- `LstEraseList`
- `LstGetNumberOfItems`
- `LstGetSelection`
- `LstGetSelectionText`
- `LstHandleEvent`
- `LstMakeItemVisible`
- `LstPopupList`
- `LstSetDrawFunction`
- `LstSetHeight`
- `LstSetListChoices`
- `LstSetPosition`
- `LstSetSelection`
- `LstSetTopItem`
- `LstSetPosition`

Menu Objects

This section provides information about menu objects by discussing these topics:

- Menu Object Overview
- Menu Events
- Associated Resources
- Menu Functions

Menu Object Overview

A menu bar is displayed when the user taps a menu icon. The menu bar, a horizontally oriented list of menu titles, appears at the top of the screen in its own window, above all the application's windows. Pressing a menu title highlights the title and "pulls down" the menu below the title.

When the user drags the pen through the menu, the command under the pen is highlighted.

- If the pen is released over a menu item, that item is selected, and the menu bar and menu disappear.
- If the pen is released outside both the menu bar and the menu, both disappear and no selection is made.
- If the pen is released in a menu title, the menu bar and the menu remain displayed until a selection is made from the menu.
- If the pen is tapped outside the menu and the menu bar, both are dismissed.

A menu has the following features:

- Item separators; lines to group menu items.
- Keyboard shortcuts; the shortcut labels are right justified in menu items.
- A menu remembers its last selection, the next time a menu is displayed the prior selection appears highlighted.
- The bits behind the menu bar and the menus are saved and restored by the menu routines.
- When the menu is visible, the insertion point is turned off.
- Selecting a separator with the pen dismisses the menu, but no event is posted.

Menu Events

Menu events are handled by the routine `MenuHandleEvent`, which handles events of type `penDownEvent` and `keyDownEvent`.

When a menu item is chosen, the menu event handler adds a `menuEvent` that identifies the chosen item to the event queue.

Structure of a Menu

```
typedef struct {
WinHandle       barWin;
WinHandle       bitsBehind;
WinHandle       savedActiveWin;
WinHandle       bitsBehindStatus;
MenuBarAttrType attr;
SWord           curMenu;
SWord           curItem;
long            commandTick;
SWord           numMenus;
MenuPullDownPtr menus;
} MenuBarType;

typedef MenuBarType * MenuBarPtr;

typedef struct {
Word visible        :1;
Word commandPending :1;
Word insPtEnabled   :1;
} MenuBarAttrType;

typedef struct {
WinHandle       menuWin;
RectangleType   bounds;
WinHandle       bitsBehind;
RectangleType   titleBounds;
CharPtr         title;
Word            numItems;
MenuItemType    *items;
} MenuPullDownType;

typedef MenuPullDownType * MenuPullDownPtr;
typedef struct {
Word    id;
char    command;
CharPtr itemStr;
} MenuItemType;
```

Menu Object Fields

The `barWin` field is the handle for the window that contains the menu bar.

The `bitsBehind` field holds a handle of a window that contains the region obscured by the menu bar.

The `savedActiveWin` field stores the currently active window behind the menu.

The `bitsBehindStatus` field stores the bits behind the status message so that when the message display terminates, the bits can be restored.

The `attr` field contains the menu bar's attributes. The `attr` field is a bit field that contains the following members: `visible`, `commandPending`, and `insPtEnabled`.

- If `visible` is set, the menu bar is drawn.
- `commandPending` is set if the next key is a command.
- `insPtEnable` is set if the insertion point was on when the menu was drawn.

The `curMenu` field holds the menu number of the currently visible menu; the menus are numbered sequentially, starting with zero. This value is preserved when the menu bar is dismissed; the next time the menu is displayed, the previously visible pull-down menu can also be redisplayed. A value of -1 indicates that there is no current pull-down menu.

The `curItem` field holds the item number of the currently highlighted menu item. The items in each menu are numbered sequentially, starting with zero.

The `commandTick` field is used to store the tick count at which the status message should be erased.

The `numMenus` field holds the number of pull-down menus on the menu bar.

The `menus` field is a pointer to an array of `MenuPullDownType` structures. The `MenuPullDownType` structure defines a pull-down menu.

Menu Pull-Down Fields

The `menuWin` field is the handle for the window that contains the menu.

The `bounds` field holds the position and size (in pixels) of the pull-down menu.

The `bitsBehind` field holds a handle of a window that contains the region obscured by the menu.

The `title` field holds a pointer to the menu title (null-terminated).

The `numItems` field holds the number of items in a menu (separators count as items).

The `items` field is a pointer to an array of `MenuItemType` structures. A `MenuItemType` structure defines a menu item.

Menu Item Fields

The `id` field holds an ID value specified by the application developer. This ID value is included as part of the event data of a `menuEvent`.

The `command` field holds the shortcut key.

The `itemStr` field holds a pointer to the text display for a menu item. The shortcut key description is included in this string. The item label and the shortcut key description are delimited with a tab character.

Associated Resources

The resources MBAR (menu bar) and MENU (menu) are used jointly to represent a menu object on screen.

Menu Functions

- `MenuDispose`
- `MenuDrawMenu`
- `MenuEraseStatus`
- `MenuGetActiveMenu`
- `MenuHandleEvent`
- `MenuInit`
- `MenuSetActiveMenu`

Date and Time UI Objects

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.

Date and Time Functions

Currently defined date and time functions `SelectDay` and `SelectTime`

Insertion Point Object

The insertion point is a blinking indicator that shows where text is inserted when users write Graffiti characters or paste clipboard text.

In general, an application does not need to be concerned with the insertion point; the Palm OS UI manages the insertion point.

Insertion Point Functions

- `InsPtEnable`
- `InsPtEnabled`
- `InsPtGetHeight`
- `InsPtGetLocation`
- `InsPtSetHeight`
- `InsPtSetLocation`

Table Objects

The table object is used to organize several types of UI objects. The number of rows and the number of columns must be specified for each table object. A UI object can be placed inside a “square” of a table. Tables often consist of rows or columns of the same object. For example, a table might have one column of labels and another column of fields. Tables can only be scrolled vertically.

This section provides information about table objects by discussing these topics:

- Table Events
- Structure of a Table
- Associated Resource
- Table Functions

Table Events

The table object generates the event `tblSelectEvent`. This event contains:

- the table’s ID number
- the row of the table selected
- the column of the table that has been selected

When `tblSelectEvent` is sent to a table, the table generates an event to handle any possible events within the item’s UI object.

Structure of a Table

```
typedef struct {
Word                    id;
RectangleType           bounds;
TableAttrType           attr;
Word                    numColumns;
Word                    numRows;
Word                    currentRow;
Word                    currentColumn;
Word                    topRow;
TableColumnAttrType *   columnAttrs;
TableRowAttrType *     rowAttrs;
TableItemPtr           items;
FieldType              currentField;
} TableType;
typedef TableType * TablePtr;

typedef struct {
Word        visible:1;
Word        editable:1;
Word        editing:1;
Word        selected:1;
} TableAttrType;

typedef struct {
TableItemStyleTypeitemType;
FontID          fontID;    //font for drawing text
Word            intValue;
CharPtr         ptr;
} TableItemType;
typedef TableItemType * TableItemPtr;

typedef struct {
Word                   width;          // in pixels
Boolean                usable;
Word                   spacing;
TableDrawItemFuncPtr   drawCallback;
TableLoadDataFuncPtr   loadDataCallback;
TabelSaveDataFuncPtr   SaveDataCallback;
} TableColumnAttrType;

typedef struct {
Word            id;
Word            height;   // row height in pixels
DWord           data;
Boolean         usable;
Boolean         selectable;
Boolean         invalid;  // true if redraw needed
} TableRowAttrType;
```

Fields of a Table Structure

The `id` field holds the ID value the application developer specified.

The `bounds` field contains the position and size of the table object.

The `attr` field contains the table object's attributes. The `attr` field is a bit field that contains the following members: `visible`, `editable`, `editing`, and `selected`.

- If a table is set to `visible`, it is drawn on the screen.
- A table is `editable` if a user can modify it.
- If the table is in edit mode, `editing` is set on.
- If the current item is selected, `selected` is set on.

The `numColumns` field specifies the number of columns in the table object.

The `numRows` field specifies the number of rows in the table object.

The `currentRow` field contains the row of the table set to current.

The `currentColumn` field contains the column of the table set to current.

The `topRow` field contains the first row in the table object.

The `columnAttrs` field contains the column's attributes, such as its width, its usability, and how to draw itself.

The `rowAttrs` field contains the row's attributes, such as its ID, height, and whether or not it is usable, selectable, or invalid.

The `items` field contains the item's attributes, such as the item type, font ID, an integer value, and a character pointer.

The `currentField` field is the field object that is currently being edited.

Associated Resource

The Table Resource (tTBL) represents a table on screen.

Table Functions

- TblDrawTable
- TblEditing
- TblEraseTable
- TblFindRowData
- TblFindRowID
- TblGetBounds
- TblGetColumnSpacing
- TblGetColumnWidth
- TblGetCurrentField
- TblGetItemBounds
- TblGetItemInt
- TblGetLastUsableRow
- TblGetNumberOfRows
- TblGetRowData
- TblGetRowHeight
- TblGetRowID
- TblGetSelection
- TblGrabFocus
- TblHandleEvent
- TblInsertRow
- TblMarkRowInvalid
- TblMarkTableInvalid
- TblRedrawTable
- TblReleaseFocus
- TblRemoveRow
- TblRowSelectable
- TblRowUsable
- TblSelectItem
- TblSetColumnSpacing
- TblSetColumnUsable
- TblSetColumnWidth
- TblSetCustomDrawProcedure

- `TblSetItemInt`
- `TblSetItemPtr`
- `TblSetItemStyle`
- `TblSetLoadDataProcedure`
- `TblSetRowData`
- `TblSetRowHeight`
- `TblSetRowID`
- `TblRowInvalid`
- `TblSetRowSelectable`
- `TblSetRowUsable`
- `TblSetSaveDataProcedure`
- `TblUnhighlightSelection`

Window Objects

A window defines a drawing region. This region may be on the display or a memory buffer (and offscreen window). Offscreen windows are useful for saving and restoring regions of the display that are obscured by other UI objects. All forms are windows, but not all windows are forms.

The window object is the portion of the form object that determines how the form's window looks and behaves. A window object contains viewing coordinates of the window and clipping bounds.

This section provides information about windows by discussing these topics:

- Window Events
- Structure of a Window
- Window Functions

Window Events

When a window becomes active, a `winEnterEvent` takes place. When the window is deactivated, a `winExitEvent` occurs. The `winEnterEvent` usually follows right after a `winExitEvent`; an old window is deactivated just before a new window is deactivated.

Structure of a Window

```
typedef struct WinTypeStruct {
Word                     displayWidth;
Word                     displayHeight;
VoidPtr                  displayAddr;
WindowFlagsType          windowFlags;
RectangleType            windowBounds;
AbsRectType              clippingBounds;
PointType                viewOrigin;
FrameBitsType            frameType;
GraphicStatePtr          gstate;
struct WinTypeStruct*    nextWindow;
} WindowType;

typedef WindowType * WinPtr;

typedef WinPtr WinHandle;
```

Fields of a Window Structure

The `displayWidth` field contains the width, in pixels, of the display memory buffer (video RAM) for onscreen windows and the width of a memory buffer for offscreen windows.

The `displayHeight` field contains the height, in pixels, of the device display.

The `displayAddr` field is a pointer to the window's display memory buffer.

The `windowFlags` field contains the window's following attributes: `format`, `offscreen`, `modal`, `focusable`, `enabled`, `visible`, `dialog`, and `compressed`.

The flags are defined as follows:

Attribute	**Set to 0**	**Set to 1**
format	screen mode	generic mode
off screen	on screen	off screen
modal	modeless window	modal window
focusable	non-focusable	focusable
enabled	disabled	enabled
visible	invisible	visible
dialog	nondialog	dialog
compressed	uncompressed	compressed

The `windowBounds` field contains the bounds of the window.

The `clippingBounds` field contains the bounds for clipping any drawing within the window.

The `viewOrigin` field is the window's origin point on the display.

The `frameType` field specifies the frame's corner diameter, width of shadow, and width of frame.

The `gstate` field specifies the state of the graphic mode, pattern mode, font, and underline mode.

The `nextWindow` field is a pointer to the next window in a linked list of windows.

Window Functions

- WinAddWindow
- WinClipRectangle
- WinCopyRectangle
- WinCreateOffscreenWindow
- WinCreateWindow
- WinDeleteWindow
- WinDisableWindow
- WinDisplayToWindowPt
- WinDrawBitmap
- WinDrawChars
- WinDrawGrayLine
- WinDrawGrayRectangleFrame
- WinDrawInvertedChars
- WinDrawLine
- WinDrawRectangle
- WinDrawRectangleFrame
- WinDrawWindowFrame
- WinDrawWindowFrame
- WinEnableWindow
- WinEraseChars
- WinEraseLine
- WinEraseRectangleFrame
- WinEraseWindow
- WinFillLine
- WinFillRectangle
- WinGetActiveWindow
- WinGetClip
- WinGetDisplayExtent
- WinGetDisplayWindow
- WinGetDrawWindow
- WinGetFirstWindow
- WinGetFramesRectangle

- `WinGetPattern`
- `WinGetWindowBounds`
- `WinGetWindowExtent`
- `WinGetWindowFrameRect`
- `WinGetWindowPointer`
- `WinInitializeWindow`
- `WinInvertChars`
- `WinInvertLine`
- `WinInvertRectangle`
- `WinInvertRectangleFrame`
- `WinModal`
- `WinRemoveWindow`
- `WinResetClip`
- `WinRestoreBits`
- `WinSaveBits`
- `WinScrollRectangle`
- `WinSetActiveWindow`
- `WinSetClip`
- `WinSetDrawWindow`
- `WinSetPattern`
- `WinSetUnderlineMode`
- `WinWindowToDisplayPt`

6

Using Palm OS Managers

In contrast to desktop computer operating systems, Palm OS consists of only one library. This library, however, contains several managers, which are groups of functions that work together to implement certain functionality. As a rule, all functions that belong to one manager use the same three-letter prefix and work together to implement a certain aspect of functionality.

In this chapter, you learn about all Palm OS managers that aren't directly responsible for memory management:

- The System Manager is responsible for the basic operation of the system. Discussion of the system manager includes information on System Boot and Reset, Power Management, The Microkernel, and Application Support.
- The Feature Manager provides information about the system software version, and what optional system features and third party extensions are installed. An application can also use the feature manager to keep track of its own data.
- The String Manager is a set of string manipulation functions available to applications. Use these routines instead of the standard C routines.
- The Time Manager provides real-time clock functions and system tick functions.
- The System Event Manager provides an interface to the low-level pen and key event queues, translates taps on silk-screened icons into key events, sends pen strokes in the Graffiti area to the Graffiti recognizer, and puts the system into low-power doze mode when there is no user activity.
- The Pen Manager provides an interface to the digitizer hardware and converts input from the digitizer into pen coordinates. Most applications never need to call the pen manager directly because any pen activity is automatically returned to the application in the form of events.
- The Key Manager provides an interface to the hardware buttons on the Palm OS device. It converts hardware button

presses into key events and implements auto-repeat of the buttons. Most applications never need to call the key manager directly except to change the key repeat rate or poll the current state of the keys.

- The Graffiti Manager provides an interface to the Graffiti recognizer. The recognizer converts pen strokes into key events which are fed to an application through the event manager. Most applications never need to call the Graffiti manager directly because the event manager calls it automatically whenever it detects pen strokes in the Graffiti area of the digitizer.
- The Alarm Manager provides support for setting real-time alarms to perform some periodic activity or display a reminder.
- The Alert Manager lets applications implement modal dialog boxes that display an alert dialog or prompt the user for a response to a question.
- The Sound Manager lets applications and system modules control sound manager settings and play custom and predefined system sounds.
- The Error Manager can be used by applications or system software for displaying unexpected run-time errors, such as those that typically show up during program development. It is not anticipated that final production versions of applications or system software will use the error manager.

The System Manager

The Palm OS system manager is responsible for the general operation of the system, including boot-up, power-up, launching applications, library management, monitoring the battery, multitasking, timing, and semaphore support. Applications need to be concerned with very few system manager API functions. Most of what the system manager does is transparent to applications and is explained here as background information only.

In this section, you learn about the following aspects of the system manager:

- System Boot and Reset provides information about the different reset operations.
- Power Management describes the three different power modes and provides guidelines for application developers.
- The Microkernel briefly describes the basic task management provided by the system.
- Application Support discusses event processing and interapplication communication from the system's point of view.
- Using the System Manager provides an overview of the API available for accessing the functionality described in the other sections. Links to the actual function descriptions are included.
- System Manager Functions lists all system manager functions available to applications for easy access.

System Boot and Reset

The system manager provides support for booting the Palm OS device. Booting occurs only when the user presses the reset switch on the device (see `Palm OS Device Reset Switch`.) Palm OS differs from a traditional desktop system in that it is never really turned off. Power is constantly supplied to essential subsystems and the on/off key is merely a way of bringing the device in or out of low-power mode (see Palm OS Power Modes). The obvious effect of pressing the on/off key is that the LCD turns on or off. When the user presses the power key to turn the device off it disables the LCD, which makes it appear as if power to the entire unit is turned off. In fact, the memory system, real-time clock, and interrupt generation circuitry are still running, though they are consuming little current.

In this first version of Palm OS, there is only one user interface application running at a time. The User Interface Application Shell (UIAS) is responsible for managing the current user-interface application. The UIAS launches the current user-interface application as a subroutine and does not get control back until that application quits. When control returns to the UIAS, it immediately launches the next application as another subroutine. See The Microkernel for more information.

Power Management

This section looks at Palm OS power management, discussing Palm OS Power Modes and Guidelines for Application Developers

See Power Management Calls for an overview of the API.

Palm OS Power Modes

At any time, the Palm OS device is in one of three power modes: sleep, doze, or running. The system manager controls transitions between different power modes and provides an API for controlling some aspects of the power management.

- **Sleep mode**. If the unit appears to be off, it is actually in sleep mode and is consuming as little current as possible. At this rate, a unit could sit for almost a year on a single set of batteries without losing the contents of memory. To enter sleep mode, the system puts as many peripherals as possible into low-power mode and sets up the hardware so that an interrupt from any hard key or the real-time clock wakes up the system. When the system gets one of these interrupts while in sleep mode, it quickly checks that the battery is strong enough to complete the wake-up and then takes each of the peripherals, for example the LCD, serial port, and timers, out of low-power mode.

  The system reenters sleep mode when the user presses the on/off key again, when the system has been idle for the minimum auto-off time, or when the battery level reaches a critically low level.
- **Doze mode**. In doze mode, the processor is halted, but all peripherals including the LCD are powered up. The system can come out of doze mode much faster than it can come out of sleep mode since none of the peripherals need to be woken up. In fact, it takes no longer to come out of doze mode than to process an interrupt. Usually, when the system appears on, it is actually in doze mode and goes into running mode only for short periods of time to process an interrupt or respond to user input like a pen tap or key press.
- **Running mode**. Running means that the processor is executing instructions and all peripherals are powered up. A typical application puts the system into running mode only about 5% of the time.

Guidelines for Application Developers

Normally, applications don't need to be aware of power management except for a few simple guidelines. When an application calls `EvtGetEvent` to ask the system for the next event to process, the system automatically puts itself into doze mode until there is an event to process. As long as an application uses `EvtGetEvent`, power management occurs automatically. If there has been no user input for the amount of time determined by the current setting of the auto-off preference, the system automatically enters sleep mode without intervention from the application.

Applications should avoid providing their own delay loops. Instead, they should use `SysTaskDelay`, which puts the system into doze mode during the delay to conserve as much power as possible. Or, if an application needs to perform periodic work, it can pass a time out to `EvtGetEvent`; this forces the unit to wake up out of doze mode and to return to the application when the time out expires, even if there is no event to process. Using these mechanisms provides the longest possible battery life.

The Microkernel

Palm OS has a preemptive multitasking kernel which provides basic task management.

Most applications don't need the microkernel services because they are handled automatically by the system. This functionality is provided mainly for internal use by the system software or for certain special purpose applications.

The User Interface Application Shell (UIAS) is responsible for managing the current user-interface application. Because memory available to the system is limited, only one user-interface application runs at any time. The UIAS launches the current user-interface application as a subroutine and does not get control back until that application quits. When control returns to the UIAS, it immediately launches the next application as another subroutine.

Usually, the UIAS is the only task running. Occasionally though, an application launches another task as a part of its normal operation. One example of this is the Sync application, which launches a second task to handle the serial communication with the desktop. By creating a second task dedicated to the serial communication and giving this task a lower priority than the main user-interface task, the Sync application can provide optimal performance over the serial port without sacrificing response time to the user-interface controls.

Normally, there is no user interaction during a sync, so that the serial communication task gets all of the processor's time. But if the user does tap on the screen, for example, to cancel the sync, the user-interface task immediately processes the tap, since it has a higher priority. Alternatively, the Sync application could have been written to use just one task but then it would have to periodically poll for user input during the serial communication, which would hamper performance and user interface response time.

Application Support

The system manager provides application support in several functional areas. The following aspects of application support are discussed in this section:

- Launching and Clean-Up
- Event Processing
- Inter-Application Communication

Launching and Clean-Up

Usually, applications on the Palm OS device are launched when the user presses one of the buttons on the case or selects an application icon from the application launcher screen. Alternatively, an application can programmatically launch another application using the system manager function `SysAppLaunch`.

When the current user-interface application quits, the system manager cleans up by deleting any chunks in the dynamic heap(s) that the application left around and closing any databases left open. Note, however, that applications should perform this kind of cleanup themselves.

Event Processing

The system manager provides the infrastructure for event generation and also contains the support for handling most system-related events. Hardware activity, such as taps on the digitizer and key presses, is interpreted by interrupt handlers of the system manager and converted into events that are eventually sent to the application through the `EvtGetEvent` call. In addition, many events returned by `EvtGetEvent` are system-related events which can be processed by the system manager call `SysHandleEvent`.

Events in Palm OS include hardware- and software-generated events. Hardware-generated events are those caused directly by user interaction with the device, such as tapping on the screen with the pen, or pressing a hardware button. Software-generated events are generated by the system software as a side effect of a user interaction. These include events such as the quit event that causes an application to exit, or keyboard events generated by the Graffiti recognizer. Software-generated events can even be defined and generated by an application for its own use.

When an application calls `EvtGetEvent`, the event manager checks a number of system-event data structures and returns an event record to the application with information about the highest-priority event that needs processing. Events in Palm OS are stored in one of three event queues: a key queue, a pen queue, or a software event queue. The event queues are circular buffers containing event records stored in a first in, first out (FIFO) sequence.

Hardware-related events are typically posted by interrupt routines and include pen-downs, pen-ups (optionally including stroke data), and hard button presses. All pen-generated events are stored in the pen queue and all hard button press events are stored in the key queue. Software events are typically posted as the result of a system call and include application-quit events, window enter and exit events, user-interface control events, etc. These types of events are stored in the software event queue.

When `EvtGetEvent` is called by the application, it first checks whether any events are in the software event queue and returns the topmost event if so. If the software event queue is empty, it checks the key and pen queues. The result is that all software events generated by a particular hardware event are processed before the next hardware event is processed. For example, a pen-down hardware event may trigger the system software to generate window exit and window enter software events. Both events are then pulled from the software event queue and processed before the next hardware event is processed.

Some event types returned by `EvtGetEvent` are not actually posted into the event queue, but are artificially generated by `EvtGetEvent` when all event queues are empty. One example is the pen-moved event, which is returned if no other events are in the queues and the pen has moved since the last time `EvtGetEvent` was called. In this way, the application is notified of low-priority events such as pen movements without cluttering the event queue with them.

Hardware-generated events include pen-down, pen-up, and hard button presses. These types of events are posted into their appropriate event queue by interrupt routines. The interrupt routine for handling keyboard presses immediately enqueues the keyboard event into the key queue and sets up a periodic interrupt routine to watch for auto-repeat and key debouncing.

Software events include window enter and exit events, application quit events, and user-interface object events like control enter, control exit, etc. These events are typically generated as a side effect of a hardware-generated event like a pen-down. Software can, however, also generate key events, usually as a result of recognizing a Graffiti stroke or a tap on a silk-screened icon.

Software-generated events are posted into the appropriate event queue, but are not typically posted at interrupt time. Many of these events are inserted into the event queue by the various user-interface managers. Others, like key events, are posted by `SysHandleEvent` after recognizing a Graffiti stroke or a tap on a silk-screened icon.

In a typical application, `SysHandleEvent` is called immediately after `EvtGetEvent`. If `EvtGetEvent` returns a pen-up event in the Graffiti writing area, `SysHandleEvent` calls the Graffiti recognizer with the pen stroke information obtained from the pen queue and uses the results of the Graffiti recognizer to post one or more keyboard events into the key queue. A similar process occurs for pen-up events detected over a silk-screened icon. `SysHandleEvent` converts the pen-up to a keyboard event with a virtual key code representing the silk-screened icon.

Inter-Application Communication

The system manager provides the API for inter-application communication. This API permits any application or system routine to send an **action code** to any other application and get results back. An application that wishes to work with the global find, for example, must support the find action code.

Sending an action code to another application is like calling a specific subroutine in that application: the application responding to the action code is responsible for determining what to do given the action code constant passed on the stack as a parameter. Predefined action codes are listed in Table 2.1 on page 50 and can be found in SytemMgr.h. All the parameters for an action code are passed in a single parameter block and the results are returned in the same parameter block. How Action Codes Control the Application describes action codes in more detail.

Using the System Manager

This section provides information about the system manager routines available for the different services, discussing these topics:

- System Reset Calls
- Power Management Calls
- Application Utilities

System Reset Calls

Applications can call `SysReset` to reset the device. This call does a soft reset and has the same effect as pressing the reset switch on the unit. **Normally, applications should not use this call**. One example of where this function is used is the Sync application. When the user copies an extension onto the Palm OS device, the Sync application automatically resets the device after the sync is completed to allow the extension to install itself.

A similar, but even more dangerous call is `SysColdBoot` which performs a hard reset that clears all user storage RAM on the device, destroying all user data.

Power Management Calls

The system calls SysSleep to put itself immediately into low-power sleep mode. Normally, the system puts itself to sleep when there has been no user activity for the minimum auto-off time or when the user presses the power key.

The SysSetAutoOffTime routine changes the auto-off time value. This routine is normally used by the system only during boot, and by the Preferences application. The Preferences application saves the user preference for the auto-off time in a preferences database and the system initializes the auto-off time to the value saved in the preferences database during boot. While the auto-off feature can be disabled entirely by calling SysSetAutoOffTime with a time out of 0, doing this depletes the battery.

The current battery level and other information can be obtained through the SysBatteryInfo routine. This call returns information about the battery, including the current battery voltage in hundredths of a volt, the warning thresholds for the low-battery alerts, the battery type, and whether or not external power is applied to the unit. This call can also change the battery warning thresholds and battery type.

Application Utilities

The SysHandleEvent call allows applications to correctly respond to system events like key presses, Graffiti strokes, low-battery warnings, and taps on silk-screened icons. This routine should be called from every application's event loop, usually before the application even looks at the event. If an application needs to override any part of the default system behavior, it could selectively filter out events before calling SysHandleEvent.

An application can force a switch to another user-interface application by calling SysUIAppSwitch. This routine notifies the system which application to launch next and feeds an application-quit event into the event queue. If and when the current application responds to the quit event and returns, the system launches the new application.

The routine SysCurAppDatabase can be used to get the card number and database ID of the currently running user-interface application. Note that if your application code is being called to process an action code, it essentially is being called as a subroutine from the current user-interface application and this routine does not return your application's database ID but the database ID of the application that initiated the action code.

The routine `SysAppLaunch` is a general-purpose launch facility for launching any resource database with executable code in it. It has numerous options, including whether or not to launch the database as a separate task or not, whether to allocate a globals world, and whether or not to give the database its own stack. This routine is also used to send action codes to applications (by telling it to use the caller's stack, no globals world, and not a separate task). Usually, applications use it only for sending action codes to other user-interface applications. An alternative, simpler method of sending action codes is the `SysBroadcastActionCode` call. This routine automatically finds all other user-interface applications and calls `SysAppLaunch` to send the action code to each of them.

System Manager Functions

The following system manager functions are available for application use:

- `SysReset`
- `SysBatteryInfo`
- `SysSetAutoOffTime`
- `SysHandleEvent`
- `SysUIAppSwitch`
- `SysCurAppDatabase`
- `SysBroadcastActionCode`
- `SysAppLaunch`

The Feature Manager

The feature manager can be used by applications to determine what the system environment is like. It provides information about the system software version, what optional system features are installed, and what third party extensions are installed. In addition, it can be used privately by an application for keeping track of its own data.

This section introduces the feature manager by discussing these topics:

- Feature Manager Overview
- Using the Feature Manager
- Feature Manager Functions

Feature Manager Overview

A **feature** is a 32-bit value that has special meaning to both the feature publisher and to users of that feature. Features can be published by the system or by applications. Each feature is identified by a feature creator and a feature number. The feature creator is usually the database creator type of the application that publishes the feature. The feature number is any 16-bit value used to distinguish between different features of a particular creator. Once a feature is published, it remains present until it is explicitly deleted. A feature published by an application sticks around even after the application quits.

One example is the feature the system publishes that contains a 32-bit representation of the system version. This is a feature with a feature creator of “psys” and a feature number of 1. The first version of the Palm OS system software has a value of 0x01003001 in this feature. Any application can find out the system version by looking for this feature.

When an application adds or removes capabilities from the base system, it can create features to test for the presence or absence of those capabilities. This allows an application to be compatible with multiple versions of the system by refining its behavior depending on which capabilities are present or not. Future hardware platforms may lack some capabilities present in the first platform, so checking the system version feature is important.

Applications may find the feature manager useful for their own private use. For example, an application may want to publish a feature that contains a pointer to some private data it needs for processing action codes. Because an application's global data is not generally available while it processes action codes, using the feature manager is usually the easiest way for an application to get to its data.

Using the Feature Manager

To check whether a particular feature is present, call `FtrGet` and pass it the feature creator and feature number. If the feature exists, `FtrGet` returns the 32-bit value of the feature. If the feature does not exist, an error code is returned.

To publish a new feature or change the value of an existing one, call `FtrSet` and pass the feature creator and number, and the 32-bit value of the feature. A published feature remains available until it is explicitly removed by a call to `FtrUnregister` or until the system resets; simply quitting an application does not remove a feature published by that application.

Features are split into two groups: ROM-based and RAM-based. ROM-based features are stored in a separate table in ROM and can never be removed; only system-defined features are in this table. All features installed at run-time are in the RAM table. `FtrGetByIndex` accepts a parameter that specifies whether to search the ROM- or RAM-table.

Call `FtrUnregister` to remove RAM-based features created at run-time by calling `FtrSet`.

You can get a complete list of all published features by calling `FtrGetByIndex` repeatedly. Passing an index value starting at 0 to `FtrGetByIndex` and incrementing repeatedly by 1 eventually returns all available features.

Feature Manager Functions

The following feature manager functions are for application use:

- `FtrGet`
- `FtrSet`
- `FtrUnregister`
- `FtrGetByIndex`

The String Manager

The string manager provides a set of string manipulation functions. The string manager API is closely modeled after the standard C string-manipulation functions like `strcpy`, `strcat`, etc.

Applications should use the functions built into the string manager instead of the standard C functions, because doing so makes the application smaller. When your application uses the string manager functions, the actual code that implements the function is not linked into your application but is already part of the operating system. When you use the standard C functions, however, the code for each function you use is linked into your application and results in a bigger executable. In addition, many standard C function don't work on the Palm OS device at all because the OS doesn't provide all basic system functions (such as `malloc`) and doesn't support the subroutine calls used by most standard C functions.

The following functions are available for application use:

- `StrCopy`
- `StrCat`
- `StrLen`
- `StrCompare`
- `StrCaselessCompare`
- `StrToLower`
- `StrIToA`
- `StrIToH`
- `StrChr`
- `StrStr`
- `StrAToI`

The Time Manager

This section first provides an overview of the date and time manager (called time manager in this chapter) and then discusses these topics:

- Using Real-Time Clock Functions
- Using System Ticks Functions
- Time Manager Function Summary

The time manager provides access to both the 1-second and 0.01-second timing resources on the Palm OS device.

- The -second timer keeps track of the real-time clock (date and time), even when the unit is in sleep mode.
- The 0.01-second timer, also referred to as the **system ticks**, can be used for finer timing tasks. This timer is not updated when the unit is in sleep mode and is reset to 0 each time the unit resets.

The basic time-manager API provides support for setting and getting the real-time clock in seconds and for getting the current system ticks value (but not for setting it). The system manager provides more advanced functionality for setting up a timer task which executes periodically or in a given number of system ticks.

Using Real-Time Clock Functions

The real-time clock functions of the time manager include `TimSetSeconds` and `TimGetSeconds`. Real time on the Palm OS device is measured in seconds from midnight, Jan 1, 1904. Call `TimSecondsToDateTime` and `TimDateTimeToSeconds` to convert between seconds and a structure specifying year, month, day, hour, minute, and second.

Using System Ticks Functions

The Palm OS device maintains a tick count that starts at 0 when the device is reset. This tick increments

- 100 times per second when running on the Palm OS device
- 60 times per second when running on the Macintosh under the simulator

For tick-based timing purposes, applications should use the macro `sysTicksPerSecond`, which is conditionally compiled for different platforms.

Use the function `TimGetTicks` to read the current tick count.

Although the `TimGetTicks` function could be used in a loop to implement a delay, it is recommended that applications use the `SysTaskDelay` function instead. The `SysTaskDelay` function automatically puts the unit into low-power mode during the delay. Using `TimGetTicks` in a loop consumes much more current.

Time Manager Structures

The time manager uses these structures to store information:

```
typedef struct{
Sword   second;
Sword   minute;
Sword   hour;
Sword   day;
Sword   month;
Sword   year;
Sword   weekDay;      //Days since Sunday (0 to 6)
}DateTimeType;

typedef DateTimeType* DateTimePTr;

typedef struct {
Byte    hours;
Byte    minutes;
}TimeType;
```

```
typedef TimeType * TimePtr;

typedef struct{
Word year    :7; //years since 1904 (Mac format)
Word month   :4;
Word day     :5;
}DateType;

typedef DateType * DatePtr;
```

Time Manager Function Summary

- DateAdjust
- DateDaysToDate
- DateSecondsToDate
- DateToAscii
- DateToDays
- DateToDOWDMFormat
- DayOfMonth
- DayOfWeek
- DaysInMonth
- TimAdjust
- TimDateTimeToSeconds
- TimGetSeconds
- TimGetTicks
- TimSecondsToDateTime
- TimSetSeconds
- TimeToAscii

The System Event Manager

The system event manager manages the low-level pen and key event queues, translates taps on silk-screened icons into key events, sends pen strokes in the Graffiti area to the Graffiti recognizer, and puts the system into low-power doze mode when there is no user activity. Most applications have no need to call the system event manager directly because most of the functionality they need comes from the higher-level event manager or is automatically handled by the system.

Applications that do use the system event manager directly might do so to enqueue key events into the key queue, or to retrieve each of the pen points that comprise a pen stroke from the pen queue. This section provides information about the system event manager by discussing these topics:

- Event Translation: Pen Strokes to Key Events
- Pen Queue Management
- Auto-Off Control
- System Event Manager Function Summary

Event Translation: Pen Strokes to Key Events

One of the higher-level functions provided by the system event manager is converting pen strokes on the digitizer to key events. For example, the system event manager sends any stroke in the Graffiti area of the digitizer automatically to the Graffiti recognizer for conversion to a key event. Taps on silk-screened icons, such as the application launcher, Menu button, and Find button, are also intercepted by the system event manager and converted into the appropriate key events.

The basic process of converting a pen stroke to key event involves:

- getting all the pen points that comprise the stroke out of the pen queue
- converting the stroke into the matching key event
- enqueuing that key event into the key queue

Eventually, the key event is returned to the application as a normal result of calling `EvtGetEvent`.

Most applications rely on the default behavior of the system event manager, which is to take all strokes in the predefined Graffiti area of the digitizer and convert them into key events, to convert all taps on the silk-screened icons to the default key events, and to pass all other strokes on to the application for processing.

Pen Queue Management

The pen queue is a preallocated area of system memory used for capturing the most recent pen strokes on the digitizer. It is a circular queue with a first-in, first-out method of storing and retrieving pen points. Points are usually enqueued by a low-level interrupt routine and dequeued by the system event manager or application.

When the user first brings the pen down on the digitizer, the system stores a pen-down sequence in the pen queue and starts the stroke capture. Periodically, additional points are stored into the pen queue. When the user lifts the pen, the system stores a pen-up sequence in the pen queue and turns off pen capture.

The system event manager provides an API for initializing and flushing the pen queue and for queuing and dequeueing points. There is some state information stored in the queue itself: to dequeue a stroke, the caller must first make a call to dequeue the stroke information (`EvtDequeuePenStrokeInfo`) before the points for the stroke can be dequeued. Once the last point is dequeued, another `EvtDequeuePenStrokeInfo` call must be made to get the next stroke.

Applications usually needn't call `EvtDequePenStrokeInfo` because the event manager calls this function automatically when it detects a complete pen stroke in the pen queue. After calling `EvtDequePenStrokeInfo`, the system event manager stores the stroke bounds into the event record and returns the pen-up event to the application. The application is then free to dequeue the stroke points from the pen queue, or to ignore them altogether. If the points for that stroke are not dequeued by the time `EvtGetEvent` is called again, the system event manager automatically flushes them.

Key Queue Management

The key queue is an area of system memory preallocated for capturing key events. Key events come from one of two occurrences: as a direct result of the user pressing one of the buttons on the case or as a side effect of the user drawing a Graffiti stroke on the digitizer which is converted in software to a key event.

When a hardware button is pressed, an interrupt routine enqueues the appropriate key event into the key queue, temporarily disables further hardware button interrupts, and sets up a timer task to run every 10ms. This timer task is used to debounce the hardware button and to support auto-repeat of the key if the user happens to hold it down for an extended period of time. If this timer task sees that the key is released for at least a minimum amount of time, it then re-enables the hardware button interrupts.

When a pen stroke is detected in the Graffiti area of the digitizer, the system manager calls the Graffiti recognizer, which then removes the stroke from the pen queue, converts the stroke into one or more key events, and finally enqueues these key events into the key queue. Similarly, when a pen stroke is detected on one of the silk-screened icons, the system event manager converts the stroke into the appropriate key event and enqueues it into the key queue.

The system event manager provides an API for initializing and flushing the key queue and for enqueuing and dequeuing key events. Usually, applications have no need to dequeue key events since the event manager does this automatically if it detects a key in the queue and returns a `keyDownEvent` to the application through the `EvtGetEvent` call.

Auto-Off Control

Because the system event manager manages the hardware events like pen taps and hardware button presses, it is responsible for resetting the auto-off timer on the device. Every time a hardware event is detected, the auto-off timer is automatically reset to 0. If an application needs to reset the auto-off timer manually, it can do so through the system event manager call `EvtResetAutoOffTimer`.

System Event Manager Function Summary

The following functions are part of the developer API to the system event manager:

- `EvtAddEventToQueue`
- `EvtCopyEvent`
- `EvtDequeuePenPoint`
- `EvtDequeuePenStrokeInfo`
- `EvtEnableGraffiti`
- `EvtEnqueueKey`
- `EvtFlushKeyQueue`
- `EvtFlushNextPenStroke`
- `EvtFlushPenQueue`
- `EvtGetEvent`
- `EvtGetPen`
- `EvtKeyQueueEmpty`
- `EvtKeyQueueSize`
- `EvtKeyQueueEmpty`
- `EvtGetPenBtnList`
- `EvtPenQueueSize`
- `EvtProcessSoftKeyStroke`
- `EvtResetAutoOffTimer`
- `EvtWakeup`

The Pen Manager

The pen manager manages the digitizer hardware and converts input from the digitizer into pen coordinates. Most applications never need to call the pen manager directly because any pen activity is automatically returned to the application in the form of events.

Pen coordinates are stored in the pen queue as raw, uncalibrated coordinates. When the system event manager routine for removing pen coordinates from the pen queue is called, it converts the pen coordinate into screen coordinates before returning.

The Preferences application provides a user interface for calibrating the digitizer. It uses the pen manager API to set up the calibration which is then saved into the Preferences database. The pen manager assumes that the digitizer is linear in both the x and y directions; the calibration is therefore a simple matter of adding an offset and scaling the x and y coordinates appropriately.

Pen Manager Functions

- `PenResetCalibration`
- `PenCalibrate`

The Key Manager

The key manager manages the hardware buttons on the Palm OS device. It converts hardware button presses into key events and implements auto-repeat of the buttons. Most applications never need to call the key manager directly except to change the key repeat rate or to poll the current state of the keys. The event manager is the main interface to the keys since it returns a `keyDownEvent` to an application whenever a button is pressed.

Normally, applications are notified of key presses through the event manager. Whenever a hardware button is pressed, the application receives an event through the event manager with the appropriate key code stored in the event record. The state of the hardware buttons can also be queried by applications at any time through the `KeyCurrentState` function call.

The key manager also provides functions for changing the auto-repeat rate of the hardware buttons. This might be useful to game applications that want to use the hardware buttons for control. The current key repeat rates are stored in the key manager globals and should be restored before the application exits.

Key Manager Functions

- `KeyRates`
- `KeyCurrentState`

The Graffiti Manager

The Graffiti manager provides an API to the Graffiti recognizer in Palm OS. This recognizer converts pen strokes into key events, which are then fed to an application through the event manager. Most applications never need to call the Graffiti manager directly because it is automatically called by the event manager whenever it detects pen strokes in the Graffiti area of the digitizer.

Special purpose applications, such as a Graffiti tutorial, may want to call the Graffiti manager directly to recognize strokes in other areas of the screen or to customize the Graffiti behavior.

The Graffiti manager provides a high-level call used by the event manager for converting pen strokes into key events. This call, `GrfProcessStroke`, removes pen points from the pen queue, recognizes the stroke, and puts one or more key events into the key queue. This call automatically handles Graffiti ShortCuts and calls the user interface as appropriate to display shift indicators in the current window.

An application can call `GrfProcessStroke` when it gets a `penUpEvent` from the event manager if it wants to recognize strokes entered into its application area (in addition to the Graffiti area).

Other high-level calls provided by the Graffiti manager include routines for getting and setting the current Graffiti shift state (caps lock on/off, temporary shift state, etc.) and for notifying Graffiti when the user selects a different field. Graffiti needs to be notified when a field change occurs so that it can cancel out of any partially entered shortcut and clear its temporary shift state if it's showing a potentially accented character.

The remainder of Graffiti manager API routines are for special purpose use. They are basically all the entry points into the Graffiti recognizer engine and are usually called only by `GrfProcessStroke`. This includes calls to add pen points to the Graffiti recognizer's stroke buffer, to convert the stroke buffer into a Graffiti glyph ID, and to map a glyph into a string of one or more key strokes.

Other routines provide access to the Graffiti ShortCuts database. This is a separate database owned and maintained by the Graffiti manager that contains all of the shortcuts. This database is opened by the Graffiti manager when it initializes and stays open even after applications quit. The only way to modify this database is through the Graffiti manager API. It provides calls for getting a list of all shortcuts, and for adding, editing, and removing shortcuts. The ShortCuts screen of the Preferences application provides a user-interface for modifying this database.

Graffiti Manager Function Summary

- `GrfProcessStroke`
- `GrfGetState`
- `GrfSetState`
- `GrfFlushPoints`
- `GrfAddPoint`
- `GrfInitState`
- `GrfCleanState`
- `GrfMatch`
- `GrfGetMacro`
- `GrfGetAndExpandMacro`
- `GrfFilterPoints`
- `GrfGetNumPoints`
- `GrfGetPoint`
- `GrfFindBranch`
- `GrfMatchGlyph`
- `GrfGetGlyphMapping`
- `GrfGetMacroName`
- `GrfDeleteMacro`
- `GrfAddMacro`

The Alarm Manager

The Palm OS alarm manager provides support for setting real-time alarms, for performing some periodic activity, or for displaying a reminder. This section helps you use the alarm manager by discussing these topics:

- Alarm Manager Overview
- Using the Alarm Manager

Alarm Manager Overview

The alarm manager works closely with the time manager to handle real-time alarms such as those set by the DateBook for meeting reminders. The alarm manager does not actually provide reminder dialog boxes or play the alarm sound. Instead, it sends the `sysAppLaunchCmdAlarmTriggered` action code to the application to display the dialog and play the sound.

The alarm manager handles alarms by application; it can queue up only one active alarm for a given application. When the alarm goes off, the `sysAppLaunchCmdAlarmTriggered` action code notifies the application. At this time, it can set the next alarm, play a short sound, or perform some maintenance activity. Triggered alarms are queued up until the action code can be sent to the creator. However, if the alarm table becomes full, the oldest entry in the table that has been both triggered and notified is deleted to make room for a new alarm.

The alarm manager orders all pending alarms, and programs the time manager (via the `TimSetAlarm` system function) to generate an interrupt for the alarm that should go off the soonest. When this interrupt occurs, the time manager calls the alarm manager's `AlmAlarmCallback` system function, which notes the interrupt by setting the "triggered" flag in the Alarm globals and calling `EvtWakeup` to wake up the event manager in case it was asleep.

The event manager calls the `AlmDisplayAlarm` system function between events. `AlmDisplayAlarm` checks the `triggered` flag and returns immediately if no alarm had been triggered. If an alarm had been triggered, `AlmDisplayAlarm` notifies all applications to set an alarm for that alarm time via the alarm triggered action code, and then calls each in turn to display its alarm. If a new alarm time is triggered while an older alarm is displayed, all applications with alarms scheduled for that time are notified, but the display cycle is postponed until all earlier alarms finish displaying.

Using the Alarm Manager

To set an alarm for a given application, call `AlmSetAlarm`. If an untriggered alarm already exists for this application, it is replaced with the new setting. The caller must pass the following:

- The storage card number and Local ID of the application (this information is used to dispatch alarm manager notifications to the application, and as the primary key for the alarm entry in the alarm table).
- Caller-specific reference value (this can be any 32-bit value which is passed to the application with all alarm manager notifications pertaining to this alarm).
- Alarm date and time expressed in seconds since 1/1/1904 (see `TimDateTimeToSeconds` and other Date/Time functions). If the alarm date and time parameter is zero, the current active alarm for the application, if any, is cancelled.

When the active alarm is triggered, the alarm manager calls the application with the `sysAppLaunchCmdAlarmTriggered` action code. If your application needs to display a reminder or other form of UI, use the `sysAppLaunchCmdDisplayAlarm` action code, which is generated next.

To retrieve the current active alarm setting for an application, call `AlmGetAlarm`, passing the storage card number and Local ID of the application. In addition, pass a pointer to the memory location for returning the caller-specific reference value which was passed when setting the alarm with `AlmSetAlarm`. `AlmSetAlarm` returns the alarm date and time expressed in seconds since 1/1/1904. The return value is zero if no active alarm exists for the application.

Alarm Manager Function Summary

- `AlmGetAlarm`
- `AlmSetAlarm`

The Alert Manager

The alert manager provides a simple way for an application to implement modal dialog boxes that display an alert message or prompt the user for a response to a question.

Given a resource ID that defines an alert, the alert manager creates and displays a modal dialog box. When the user taps one of the buttons in the dialog, the alert manager disposes of the dialog box and returns to the caller the item number of the button the user tapped.

There are four types of alerts:

- Question
- Warning
- Notification
- Error

The alert type determines which icon is drawn in the alert window and the sound that plays when the alert is displayed.

The Alert Resource

When the alert manager is invoked, it is passed an alert resource (see Alert Resource) that contains the following information:

- The rectangle that specifies the size and position of the alert window.
- The alert type (question, warning, notification, or error).
- The null-terminated text string; that is, the message the alert displays.
- The text labels for one or more buttons.

Alert Manager Functions

- `FrmAlert`
- `FrmCustomAlert`

The Sound Manager

The Palm OS sound manager lets applications and system modules play custom and predefined system sounds and control sound manager settings.

The sound manager provides an extendable API for playing custom sounds, system sounds, and for controlling default sound settings. Although the API accommodates multichannel design, only a single sound channel is currently supported. Current custom sound support permits the user to specify the frequency, duration, and amplitude of the sound. Currently supported system sounds are Information, Warning, Error, Start-up, Alarm, Confirmation, and Click. The user (typically the Preferences application) can control the alarm, system, and master sound amplitudes.

Using the Sound Manager

To execute a sound manager command, call `SndDoCmd` and pass the sound channel pointer (presently, only null is supported and maps to the shared channel), a pointer to a structure of `SndCommandType`, and a flag indicating whether the command should be performed asynchronously (not yet implemented; all commands execute synchronously).

To play a default system sound, such as a click or an error beep, call `SndPlaySystemSound`, passing the system sound id. For the complete list of system sound IDs, see SoundMgr.h.

Sound Manager Function Summary

- `SndDoCmd`
- `SndGetDefaultVolume`
- `SndPlaySystemSound`
- `SndSetDefaultVolume`

The Error Manager

The error manager can be used by applications or system software for displaying unexpected run-time errors such as those that typically show up during program development. Final versions of applications or system software won't use the error manager. The error manager API consists of a set of functions for displaying an alert with an error message, file name, and the line number where the error occurred. If a debugger is connected, it is entered when the error occurs.

The error manager also provides a "try and catch" mechanism that applications can use for handling such run-time errors as out of memory conditions, user input errors, etc. This mechanism is closely modeled after the try/catch functionality of the recent ANSI C specification.

Displaying Development Errors

The error manager provides some compiler macros which can be used in source code. These macros display a fatal alert dialog on the screen and provide buttons to reset the device or enter the debugger after the error is displayed. There are three macros: `ErrDisplay`, `ErrFatalDisplayIf`, and `ErrNonFatalDisplayIf`. `ErrDisplay` always displays the error message on the screen, `ErrFatalDisplayIf` and `ErrNonFatalDisplayIf` display the error message only if their first argument is TRUE.

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: full, partial, and none.

- Setting the value of the compiler define to ERROR_CHECK_NONE (0) does not compile in any calls.
- Setting the value to ERROR_CHECK_PARTIAL (1) compiles in only the `ErrDisplay` and `ErrFatalDisplayIf` calls.
- Setting the value to ERROR_CHECK_FULL (2) compiles in all three calls.

During development, it makes sense to set full error checking for early development, partial error checking during alpha and beta test periods, and no error checking for the final product. At partial error checking, only fatal errors are displayed; error conditions which are only possible are ignored under the assumption that the application developer is already aware of the condition and designed the software to operate that way.

The Try and Catch Mechanism

The “try and catch” mechanism of the error manager is closely modeled after the ANSI C try and catch standard. The error manager is aware of the machine state of the Palm OS device and is thus written to correctly save and restore its state. (The built-in try and catch of the compiler can’t be used because of machine dependence.)

Try and catch is basically a “neater” way of implementing a GoTo if an error occurs. A typical way of handling errors in the middle of a routine is to go to the end of the routine as soon as an error occurs and then have some general purpose clean-up code at the end of every routine. Errors in nested routines are even trickier because the result code from every subroutine call must be checked before continuing.

When you set up a try/catch you are providing the compiler with a place to jump to when an error occurs. You can go to that error handling routine at any time by calling `ErrThrow`. When it sees the `ErrThrow` call, the compiler performs a `goto` to your error handling code. The greatest advantage to calling `ErrThrow`, however, is for handling errors in nested subroutine calls.

Even if `ErrThrow` is called from a nested subroutine, execution immediately goes to the same error handling code in the higher-level call. The compiler and run-time environment automatically strip off the stack frames that were pushed onto the stack during the nesting process and go to the error handling section of the higher-level call. You no longer have to check for result codes after calling every subroutine; this greatly simplifies your source code and reduces its size.

Using the Error Manager

Calls to the error manager to display errors are actually compiler macros that are conditionally compiled into your program. Most of the calls take a Boolean parameter, which should be set to TRUE to display the error, and a pointer to a text message to display if the condition is true. Typically, the Boolean parameter is an in-line expression that evaluates to true if there is an error condition. As a result, both the expression that evaluates the error condition and the message text are left out of the compiled code when error checking is turned off. You can call `ErrFatalDisplayIf` or `ErrDisplay`, but using `ErrFatalDisplayIf` makes your source code look neater.

For example, assume your source code looks like this:

```
result = DoSomething();
ErrFatalDisplayIf (result < 0, “unexpected
                    result from DoSomething”);
```

With error checking turned on, this code displays an error alert dialog if the result from `DoSomething()` is less than 0. Besides the error message itself, this alert also shows the file name and line number of the source code that called the error manager. With error checking turned off, both the expression evaluation `err < 0` and the error message text are left out of the compiled code.

The same net result can be achieved by the following code:

```
result = DoSomething();
#if ERROR_CHECK_LEVEL != ERROR_CHECK_NONE
if (result < 0)
  ErrDisplay (“unexpected result from
                DoSomething”);
#endif
```

But this solution is longer and requires more work than simply calling `ErrFatalDisplayIf`. It also makes the source code harder to follow.

Using the Try and Catch Mechanism

The following example illustrates the possible layout for a a typical routine using the error manager's try and catch mechanism:

```
ErrTry {
  p = MemPtrNew(1000);
  if (!p) ErrThrow(errNoMemory);
  MemSet(p, 1000, 0);
  CreateTable(p);
  PrintTable(p);
  }

ErrCatch(err) {
  // Recover or cleanup after a failure in the
  // above Try block."err" is an int
  // identifying the reason for the failure.

  // You may call ErrThrow() if you want to
  // jump out to the next Catch block.

  // The code in this Catch block does not
  // execute if the above Try block completes
  // without a Throw.

  if (err == errNoMemory)
    ErrDisplay("Out of Memory");
  else
    ErrDisplay("Some other error");
  } ErrEndCatch
  // You must structure your code exactly as
  //above. You can't have an ErrTry without an
  //ErrCatch { } ErrEndCatch, or vice versa.
```

Any call to `ErrThrow` within the `ErrTry` block results in control passing immediately to the `ErrCatch` block. Even if the subroutine `CreateTable` called `ErrThrow`, control would pass directly to the

`ErrCatch` block. If the `ErrTry` block completes without calling `ErrThrow`, the `ErrCatch` block is not executed.

You can nest multiple `ErrTry` blocks. For example, if you wanted to do some cleanup at the end of `CreateTable` in case of error, you could put `ErrTry`/`ErrCatch` blocks in `CreateTable`, clean up in the `ErrCatch` block there first, and then call `ErrThrow` to jump to the top-level `ErrCatch`.

Error Manager Function Summary

- `ErrDisplay`
- `ErrDisplayFileLineMsg`
- `ErrFatalDisplayIf`
- `ErrNonFatalDisplayIf`
- `ErrThrow`

7

Control, Field, and Insertion Point Functions

Control Functions

CtlDrawControl

Purpose	Draw a control object (and the text in it) on screen. The control is drawn only if its `usable` attribute is TRUE.
Prototype	`void CtlDrawControl (ControlPtr ControlP)`
Parameters	ControlP Pointer to the control object to draw.
Result	Returns nothing.
Comments	Sets the `visible` attribute to TRUE.
See Also	`CtlSetUsable`, `CtlShowControl`

CtlEraseControl

Purpose	Erase a usable and visible control object and its frame from the screen.
Prototype	`void CtlEraseControl (ControlPtr ControlP)`
Parameters	ControlP Pointer to control object to erase.
Comments	Sets the `visible` attribute to FALSE.

CtlGetLabel

Purpose	Return a character pointer to a control's text label.
Prototype	`CharPtr CtlGetLabel (ControlPtr ControlP)`
Parameters	ControlP — Pointer to control object.
Result	Returns a pointer to a null-terminated string.
See Also	`CtlSetLabel`

CtlGetValue

Purpose	Return the current value (on or off) of the specified control. This function is valid only for push buttons and check boxes. The return value is undefined for other control types.
Prototype	`short CtlGetValue (ControlPtr ControlP)`
Parameters	ControlP — Pointer to a control object.
Result	Returns the current value of the control; 0 = off, 1 = on.
See Also	`CtlSetValue`

CtlHandleEvent

Purpose

Handle event in the specified control object.

Prototype

```
Boolean CtlHandleEvent ( ControlPtr ControlP,
                         EventPtr EventP)
```

Parameters

ControlP	Pointer to control object.
EventP	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` and `ctlExitEvent`, if the control ID in the event data matches the control's ID.

Comments

The control object's `usable`, `enabled`, and `visible` attributes must be TRUE. This routine handles three type of events: `penDownEvent`, `ctlEnterEvent`, and `ctlRepeatEvent`.

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 inverted.

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. This function calls `CtlEraseControl`.
Prototype	`void CtlHideControl (ControlPtr ControlP)`
Parameters	ControlP Pointer to the control object to hide.
Result	Returns nothing.
Comments	A control that is not usable doesn't draw and doesn't respond to the pen. Sets the `visible` and the `usable` attributes to FALSE.
See Also	`CtlShowControl`

CtlHitControl

Purpose	Simulate tapping a control. This function adds a `ctlSelectEvent` to the event queue.
Prototype	`void CtlHitControl (ControlPtr ControlP)`
Parameters	ControlP Pointer to a control object.
Result	Returns nothing.
Comments	Useful for testing.

CtlEnabled

Purpose	Return TRUE if the control is enabled. Disabled controls do not respond to the pen.
Prototype	`Boolean CtlEnabled (ControlPtr ControlP)`
Parameters	ControlP — Pointer to control object.
Result	Returns TRUE if enabled, FALSE if not.
See Also	`CtlSetEnabled`

CtlSetEnabled

Purpose — Set a control as enabled or disabled. Disabled controls do not respond to the pen.

Prototype

```
void CtlSetEnabled ( ControlPtr ControlP,
                     Boolean enable)
```

Parameters

ControlP	Pointer to a control object.
enable	TRUE to set enabled, FALSE to set not enabled.

Result — Returns nothing.

See Also — `CtlEnabled`

CtlSetLabel

Purpose

Set the current label for the specified control object. If the control object currently has its `usable` and `visible` attributes set to TRUE, redraw it with the new label.

Prototype

```
void CtlSetLabel ( ControlPtr ControlP,
                   CharPtr newLabel)
```

Parameters

ControlP	Pointer to a control object.
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.

The pointer passed to this function is stored in the control's data structure; the control does not make a copy of the string passed.

See Also

`CtlGetLabel`

CtlSetUsable

Purpose

Set a control usable or not usable.

Prototype

```
void CtlSetUsable (  ControlPtr ControlP,
                     Boolean usable)
```

Parameters

ControlP	Pointer to a control object.
usable	TRUE to set usable, FALSE to set not usable.

Result

Returns nothing.

Comments

Does not usually update the control.

See Also

`CtlEraseControl`, `CtlDrawControl`

CtlSetValue

Purpose	Set the current value (on or off) of the specified control. If the control is visible, it is visually updated.
Prototype	`void CtlSetValue ( ControlPtr ControlP, short newValue)`
Parameters	ControlP — Pointer to a control object. newValue — 0 = off, non-zero = on.
Result	Returns nothing.
Comments	Does not usually update the control. This function works only with push buttons and check boxes. Other controls ignore calls to this function.
See Also	`CtlGetValue`

CtlShowControl

Purpose	Set a control's `usable` attribute to TRUE and draw the control on the screen. This function calls `CtlDrawControl`.
Prototype	`void CtlShowControl (ControlPtr ControlP)`
Parameters	ControlP — Pointer to a control object.
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 Also	`CtlHideControl`

Field UI Functions

FldCalcFieldHeight

Purpose	Determine the height of a field for a string.

Prototype

```
Word FldCalcFieldHeight   (CharPtr chars,
                            Word maxWidth)
```

Parameters

chars	Pointer to a null-terminated string.
maxWidth	Maximum line width in pixels.

Result Returns total number of lines needed to draw the string passed.

FldCompactText

Purpose Compact the memory block that contains the text of the field to release any unused space.

Prototype

```
void FldCompactText (FieldPtr fld)
```

Parameters

fld	Pointer to a field object (`FieldType` data structure).

Result Returns nothing.

Comments As characters are added to the text of a field, the block that contains the text is grown. The block is expanded in chunks so that it doesn't have to expand each time a character is added. This results in some unused space in the text block.

Needs to be called by applications on field objects which edit data records in place before the field is unlocked, or at any other time when a compact field is desirable; for example, when the form is being closed.

FldCopy

Purpose	Copy the current selection to the text clipboard.
Prototype	`void FldCopy (FieldPtr fld)`
Parameters	fld Pointer to a field object (`FieldType` data structure).
Result	Returns nothing.
Comments	This function leaves the current selection highlighted. This functions replaces anything previously in the text clipboard. If there is no selection, this function does nothing.
See Also	`FldCut`, `FldPaste`

FldCut

Purpose	Copy the current selection to the text clipboard, delete the selection from the field, and redraw the field.
Prototype	`void FldCut (FieldPtr fld)`
Parameters	fld Pointer to a field object (`FieldType` data structure).
Result	Returns nothing.
Comments	Anything previously in the text clipboard is replaced by this function. If there is no selection, this function does nothing.
See Also	`FldCopy`, `FldPaste`

FldDelete

Purpose	Delete the specified range of characters from the field and redraw the field.
Prototype	`void FldDelete (FieldPtr fld, Word start, Word end)`
Parameters	fld — Pointer to the field object to delete from. start — Starting character position. end — Ending character position.
Result	Returns nothing.
See Also	`FldInsert`

FldDirty

Purpose	Return true if the field has been modified by the user since the text value was set (`FldSetText`).
Prototype	`Boolean FldDirty (FieldPtr fld)`
Parameters	fld — Pointer to a field object (`FieldType` data structure)
Result	Returns TRUE if the field has been modified by the user, FALSE if the field has not been modified.
See Also	`FldSetDirty`

FldDrawField

Purpose	Draw the text of the field. The field's `usable` attribute must be TRUE or the field won't be drawn.
Prototype	`void FldDrawField (FieldPtr fld)`
Parameters	fld Pointer to a field object (`FieldType` data structure).
Result	Returns nothing.
Comments	This function does not 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.
Prototype	`void FldEraseField (FieldPtr fld)`
Parameters	fld Pointer to a field object (`FieldType` data structure).
Result	Returns nothing.
Comments	The function does not modify the contents of the field. If the field has the focus, the blinking insertion point is turned off.
See Also	`FldDrawField`

FldFreeMemory

Purpose
Release the memory allocated to the text of a field and the word-wrapping information.

Prototype
`void FldFreeFieldMemory (FieldPtr fld)`

Parameters

fld	Pointer to a field object (`FieldType` data structure).

Result
Returns nothing.

Comments
This function releases the memory allocated to hold the text of a field (the memory block pointed to by the `text` member of the `FieldType` data structure) and the memory allocated to hold the display lines information (the memory block pointed to by the `lines` member in the `FieldType` data structure).

This function does not affect the display of the field.

FldGetAttributes

Purpose
Return the attributes of a field.

Prototype
```
void FldGetAttributes (  FieldPtr fld,
                         FieldAttrPtr attrP)
```

Parameters

fld	Pointer to a `FieldType` structure.
attrP	Pointer to `FieldAttrType`, see Field.h.

Result
Returns nothing.

See Also
`FldSetAttributes`

FldGetBounds

Purpose	Return the current bounds of a field.
Prototype	`void FldGetBounds (FieldPtr fld, RectanglePtr rect)`
Parameters	fld Pointer to a field object (`FieldType` data structure). rect Pointer to a `RectangleType` structure.
Result	Returns the field's bounds in the `RectangleType` structure reference by `bounds`.
Comments	Returns the `rect` field of the `FieldType` structure.
See Also	`FldSetBounds`

FldGetFont

Purpose	Return the ID of the font used to draw the text of a field.
Prototype	`FontID FldGetFont (FieldPtr fld)`
Parameters	fld Pointer to a field object (`FieldType` data structure).
Result	Returns the ID of the font.
See Also	`FldSetFont`

FldGetInsPtPosition

Purpose	Return the string position of the insertion point.
Prototype	`Word FldGetInsPtPosition (FieldPtr fld)`
Parameters	fld Pointer to a field object (`FieldType` data structure).
Result	Returns the character position of insertion point.
Comments	The insertion point position number is to the left of the string position number. In multiline fields, line feeds are counted as part of the string and the position number after the line feed is the beginning of the next line.
See Also	`FldSetInsPtPosition`

FldGetMaxChars

Purpose	Return the maximum number of characters the field accepts.
Prototype	`Word FldGetMaxChars (FieldPtr fld)`
Parameters	fld Pointer to a field object (`FieldType` data structure).
Result	Returns the maximum number of characters the user is allowed to enter.
See Also	`FldSetMaxChars`

FldGetScrollPosition

Purpose	Return the string position of the first character in the first line of a field.
Prototype	`Word FldGetScrollPosition (FieldPtr fld)`
Parameters	fld Pointer to a field object (`FieldType` data structure).
Result	Returns the character position of first visible character.
See Also	`FldSetScrollPosition`

FldGetSelection

Purpose Return the current selection of a field.

Prototype

```
void FldGetSelection ( FieldPtr fld,
                       WordPtr startPosition,
                       WordPtr endPosition)
```

Parameters

fld	Pointer to a field object (`FieldType` data structure).
startPosition	Pointer to start-character position of selected range of characters.
endPosition	Pointer to end-character position of selected range of characters.

Result Returns the start and end position in `startPosition` and `endPosition`.

Comments The first character in a field is at position zero.

If the user has selected the first five characters of a field, `startPosition` will contain the value 0 and `endPosition` the value 5.

See Also `FldSetSelection`

FldGetTextAllocatedSize

Purpose	Return the number of characters allocated to hold the field's text string. Don't confuse this number with the length of the text string.
Prototype	`Word FldGetTextAllocatedSize (FieldPtr fld)`
Parameters	fld Pointer to a field object.
Result	Returns the number of characters allocated for the field's text.
See Also	`FldSetTextAllocatedSize`

FldGetTextHandle

Purpose	Return a handle to the block that contains the text string of a field.
Prototype	`Handle FldGetTextHandle (FieldPtr fld)`
Parameters	fld Pointer to a field object (`FieldType` data structure).
Result	Returns the handle of the text string of a field; 0 is a possible value.
Comments	If 0 is returned, no handle has been allocated for the field pointer.
See Also	`FldSetTextHandle`, `FldGetTextPtr`

FldGetTextHeight

Purpose	Return the number of lines of text that the specified field has.
Prototype	`Word FldGetTextHeight (FieldPtr fld)`
Parameters	fld Pointer to a field object (`FieldType` data structure).
Result	Returns the number of lines with text.
Comments	Empty lines are not counted.
See Also	`FldCalcFieldHeight`

FldGetTextLength

Purpose	Return the length of the text string of a field object.
Prototype	`Word FldGetTextLength (FieldPtr fld)`
Parameters	fld Pointer to a field object (`FieldType` data structure).
Result	Returns the length of a field's text string.

FldGetTextPtr

Purpose	Return a pointer to the text string of a field or null.
Prototype	`CharPtr FldGetTextPtr (FieldPtr fld)`
Parameters	fld Pointer to a field object (`FieldType` data structure).
Result	Returns a pointer to the text string of a field; NULL is a possible value.
See Also	`FldSetTextPtr`, `FldGetTextHandle`

FldGetVisibleLines

Purpose	Return the number of lines that can be displayed within the visible bounds of the field.
Prototype	`Word FldGetVisibleLines (FieldPtr fld)`
Parameters	fld Pointer to a field object (`FieldType` data structure).
Result	Returns the number of lines.

FldGrabFocus

Purpose	Turn the insertion point on (if the specified field is visible) and position the blinking insertion point in the field.
Prototype	`void FldGrabFocus (FieldPtr fld)`
Parameters	fld Pointer to a field object (`FieldType` data structure).
Result	Returns nothing.
Comments	This function sets the field attribute `hasFocus` to TRUE.
See Also	`FldReleaseFocus`

FldHandleEvent

Purpose	Handles the following events: `keyDownEvent`, `penDownEvent`, and `fldEnterEvent`. The field's `editable` and `usable` attributes must be set to TRUE.

Prototype

```
Boolean FldHandleEvent ( FieldPtr fld,
                         EventPtr EventP)
```

Parameters

fld	Pointer to a field object (`FieldType` data structure).
EventP	Pointer to an event (`EventType` data structure).

Result

Returns TRUE if the event was handled.

Comments

When a `keyDownEvent` occurs, the keystroke appears in the field if it is a printable character or manipulates the insertion point if it is a "movement" character. The field is automatically updated.

When a `penDownEvent` occurs, an "editable" field sends a `fldEnterEvent` to the event queue.

When a `fldEnterEvent` occurs, the field grabs the focus and the insertion point is placed in the specified position.

If the event alters the contents of the field, this function visually updates the field.

This function does not handle any events if the field is not editable.

FldInsert

Purpose: Replace the current selection with the string passed.

Prototype:
```
Boolean FldInsert ( FieldPtr fld,
                    CharPtr insertChars,
                    Word insertLen)
```

Parameters:

fld	Pointer to the field object to insert to.
insertChars	Text string to be inserted.
insertLen	Length of the text string to be inserted.

Result: Returns TRUE if the string was successfully inserted, otherwise FALSE.

Comments: If there is no current selection, the string passed is inserted at the position of the insertion point.

See Also: `FldPaste`, `FldDelete`, `FldCut`, `FldCopy`

FldMakeFullyVisible

Purpose: Cause a dynamically resizable field to expand its height to make its text fully visible.

Prototype:
```
Boolean FldMakeFullyVisible (FieldPtr fld)
```

Parameters:

fld	Pointer to a field object.

Result: Returns TRUE if the field was not fully visible, otherwise FALSE.

Comments: If the field's height changes, this function sends a `fldHeightChangedEvent` via the event queue.

Caveats: If the field is in a table, the table resizes it; otherwise, it is not resized.

FldPaste

Purpose
Replace the current selection in the field with the contents of the text clipboard.

Prototype

```
void FldPaste (FieldPtr fld)
```

Parameters

fld	Pointer to a field object (`FieldType` data structure).

Result
Returns nothing

Comments
This insertion point is positioned after the last character inserted and the field is scrolled, if necessary, so the insertion point is visible.

If there is no current selection, the clipboard text is inserted at the position of the insertion point. If there is no text in the clipboard, do not delete the current selection.

See Also
`FldInsert`, `FldDelete`, `FldCut`, `FldCopy`

FldRecalculateField

Purpose
Update the structure that contains the word-wrapping information for each visible line.

Prototype

```
void FldRecalculateField ( FieldPtr fld,
                           Boolean redraw)
```

Parameters

fld	Pointer to a field object (`FieldType` data structure).
redraw	If TRUE, redraws the field.

Result
Returns nothing.

Comments
If necessary this function reallocates the memory block that contains the displayed lines information, the block pointed to by the `lines` member in the `FieldType` data structure.

Call this function if the field data structure is modified in a way that invalidates the visual appearance of the field.

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.
Prototype	`void FldReleaseFocus (FieldPtr fld)`
Parameters	fld Pointer to a field object (`FieldType` data structure).
Result	Returns nothing.
Comments	This function sets the field attribute `hasFocus` to FALSE.
See Also	`FldGrabFocus`

FldScrollable

Purpose: Return TRUE if the field is scrollable in the direction specified.

Prototype:

```
Boolean FldScrollable (  FieldPtr fld,
                         DirectionType direction)
```

Parameters:

fld	Pointer to a field object (`FieldType` data structure).
direction	“up” or “down.”

Result: Returns TRUE if the field is scrollable, FALSE otherwise.

See Also: `FldScrollField`

FldScrollField

Purpose Scroll a field up or down by the number of lines specified.

Prototype
```
void FldScrollField ( FieldPtr fld,
                      Word linesToScroll,
                      DirectionType direction)
```

Parameters

fld	Pointer to a field object (`FieldType` data structure).
linesToScroll	Number of lines to scroll.
direction	“up” or “down.”

Result Returns nothing.

Comments Can’t scroll right or left.

The field object is redrawn if it’s scrolled.

See Also `FldScrollable`

FldSendChangeNotification

Purpose Send a `fldChangedEvent` via the event queue.

Prototype
```
void FldSendChangeNotification (FieldPtr fld)
```

Parameters

fld	Pointer to a field object.

Result Returns nothing.

FldSendHeightChangeNotification

Purpose Send a `fldHeightChangedEvent` via the event queue.

Prototype
```
void FldSendHeightChangeNotification
                ( FieldPtr fld,
                  Word pos,
                  Short numLines)
```

Parameters

fld	Pointer to a field object.
pos	Character position of the insertion point.
numLines	New number of lines in the field.

Result Returns nothing.

FldSetAttributes

Purpose Set the attributes of a field.

Prototype
```
void FldSetAttributes (  FieldPtr fld,
                         FieldAttrPtr attrP)
```

Parameters

fld	Pointer to a `FieldType` structure.
attrP	Pointer to the attributes.

Result Returns nothing.

See Also `FldGetAttributes`

FldSetBounds

Purpose	Change the position and or size of a field.
Prototype	`void FldSetBounds (FieldPtr fld, RectanglePtr rect)`
Parameters	fld — Pointer to a field object (`FieldType` data structure). rect — Pointer to a `RectangleType` structure that contains the new bounds of the display.
Result	Returns nothing.
Comments	If the field is visible, the field is redrawn within its new bounds. The memory block that contains the word-wrapping information will be resized if the number of visible lines is changed. The insertion point is assumed to be off when this routine is called.
Caveats	Don’t change the width of the object while it is visible.
See Also	`FldGetBounds`

FldSetDirty

Purpose	Set whether the field has been modified.
Prototype	`void FldSetDirty (FieldPtr fld, Boolean dirty)`
Parameters	fld — Pointer to a field object. dirty — TRUE if the text is modified.
Result	Returns nothing.
See Also	`FldDirty`

FldSetFont

Purpose: Set the font of the field, update the word-wrapping information and draw the field if the field is visible.

Prototype: `void FldSetFont (FieldPtr fld, FontID fontID)`

Parameters:

fld	Pointer to a field object (`FieldType` data structure).
fontID	ID of new font.

Result: Returns nothing.

See Also: `FldGetFont`

FldSetInsPtPosition

Purpose: Set the location of the insertion point for a given string position.

Prototype: `void FldSetInsPtPosition (FieldPtr fld, Word pos)`

Parameters:

fld	Pointer to a field object (`FieldType` data structure).
pos	Character position of insertion point.

Result: Returns nothing.

Comments: If the position is beyond the visible text, the insertion point is disabled.

See Also: `FldGetInsPtPosition`

FldSetMaxChars

Purpose	Set the maximum number of characters the field accepts.
Prototype	`void FldSetMaxChars (FieldPtr fld, Word maxChars)`
Parameters	fld — Pointer to a field object (`FieldType` data structure). maxChars — Maximum number of characters the user may enter.
Result	Returns nothing.
Comments	Line feed characters are included when the number of characters is determined.
See Also	`FldGetMaxChars`

FldSetScrollPosition

Purpose	Set the string position of the first character in the first line of a field. Redraw the field if necessary.
Prototype	`void FldSetScrollPosition (FieldPtr fld, Word pos)`
Parameters	fld — Pointer to a field object (`FieldType` data structure). pos — Character position of first visible character.
Result	Returns nothing.
See Also	`FldGetScrollPosition`

FldSetSelection

Purpose

Set the current selection in a field and highlight the selection if the field is visible.

Prototype

```
void FldSetSelection ( FieldPtr fld,
                       Word startPosition,
                       Word endPosition)
```

Parameters

fld	Pointer to a field object (`FieldType` data structure)
startPosition	Starting character position of the character range to highlight.
endPosition	End character position of the character range to highlight.

Result

Returns nothing.

Comments

This function does not affect the display; the highlight is not redrawn until the field is redrawn.

To cancel a selection, set both `startPosition` and `endPosition` to the same value.

If `startPosition` equals `endPosition`, the current selection is unhighlighted.

FldSetText

Purpose

Set the text value of the field, update the word-wrapping information, and place the insertion point after the last visible character.

Prototype

```
void FldSetText ( FieldPtr fld,
                  VoidHand textHandle,
                  Word offset,
                  Word size)
```

Parameters

fld	Pointer to a field object (`FieldType` data structure).
textHandle	Handle of a block containing a null-terminated text string.
offset	Offset from start of block to start of the text string.
size	Allocated size of text string, **not** the string length.

Result

Returns nothing.

Comments

The pointer passed is stored in the field's structure; in other words this function does not make a copy of the string passed.

If a size of zero is passed, the size is computed as the block size, less the offset passed. If more text is set than there is room for in memory, an error occurs.

WARNING: This routine does not free the memory block that holds the current text value.

See Also

`FldSetTextPtr`, `FldSetTextHandle`

FldSetTextAllocatedSize

Purpose
Set the number of characters allocated to hold the field's text string. Don't confuse this with the length of the text string.

Prototype
```
void FldSetTextAllocatedSize ( FieldPtr fld,
                               Word allocatedSize)
```

Parameters

fld	Pointer to a field object.
allocatedSize	Number of characters to allocate for the text.

Result
Returns nothing.

See Also
`FldGetTextAllocatedSize`

FldSetTextHandle

Purpose
Set the handle of the block that contains the text string of a field.

Prototype
```
void FldSetTextHandle (  FieldPtr fld,
                         Handle textHandle)
```

Parameters

fld	Pointer to a field object (`FieldType` data structure).
textHandle	Handle of a field's text string; 0 is a possible value.

Result
Returns nothing.

See Also
`FldSetTextPtr`, `FldSetText`

FldSetTextPtr

Purpose	Set the field's text to point to a text string.
Prototype	`void FldSetTextPtr (FieldPtr fld, CharPtr textP)`
Parameters	fld — Pointer to a field object (`FieldType` data structure). textP — Pointer to a null-terminated string.
Result	Returns nothing.
Comments	Since the field cannot resize a pointer (only handles can be re-sized), the field must be not editable; if the field is editable, an error occurs. This function does **not** visually update the field.
See Also	`FldSetTextPtr`, `FldSetTextHandle`

FldSetUsable

Purpose	Set a field usable or nonusable.
Prototype	`void FldSetUsable (FieldPtr fld, Boolean usable)`
Parameters	fld — Pointer to a FieldType structure. usable — TRUE to set usable, FALSE to set nonusable.
Result	Returns nothing.
Comments	A nonusable field does not display or accept input.
See Also	`FldEraseField`, `FldDrawField`

FldUndo

Purpose	Undo the last change made to the field object. Changes include typing, backspaces, delete, paste, and cut.
Prototype	`void FldUndo (FieldPtr fld)`
Parameters	fld — Pointer to the field that has the focus.
Result	Returns nothing.
See Also	`FldPaste`, `FldCut`, `FldCopy`

FldWordWrap

Purpose	Given a string and a width, return the number of characters that can be displayed using the current font.
Prototype	`Word FldWordWrap (CharPtr chars, Word maxWidth)`
Parameters	chars — Pointer to a null-terminated string. maxWidth — Maximum line width in pixels.
Result	Returns the number of characters.

Insertion Point Functions

InsPtEnable

Purpose	Enable or disable the insertion point. When the insertion point is disabled it is invisible, when it is enabled it blinks.
Prototype	`void InsPtEnable (Boolean enableIt)`
Parameters	enable TRUE = enable, FALSE = disable
Result	Returns nothing.
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 (`WinCopyRectangle`).
See Also	`InsPtEnabled`

InsPtEnabled

Purpose	Return TRUE if the insertion point is enabled or FALSE if it is disabled.
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.

Prototype `short 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.

Prototype `void InsPtGetLocation (short *x, short *y)`

Parameters

x	Pointer to top-left position of insertion point’s x coordinate.
y	Pointer to top-left position of insertion point’s 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.
Prototype	`void InsPtSetHeight (short 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 is erased and redrawn with its new height.
See Also	`InsPtGetHeight`

InsPtSetLocation

Purpose	Set the screen-relative position of the insertion point.
Prototype	`void InsPtSetLocation (short x, short y)`
Parameters	x — Number of pixels from the left side of the display. y — Number of pixels from the 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`

Functions for System Use Only

InsPtCheckBlink

Prototype `void InsPtCheckBlink (void)`

WARNING: For System Use Only.

InsPtInitialize

Prototype `void InsPtInitialize (void)`

WARNING: For System Use Only.

8

Form, List, and Menu Functions

Form Functions

FrmAlert

Purpose	Create a modal dialog from an alert resource and display it until the user selects a button in the dialog.
Prototype	`Word FrmAlert (Word 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).
See Also	`FrmDoDialog`, `FrmCustomAlert`

FrmCloseAllForms

Purpose	Sends a `frmCloseEvent` to all open forms.
Prototype	`void FrmCloseAllForms (void)`
Parameters	None.
Comments	Can be called by applications 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.

Prototype

```
void FrmCopyLabel (  FormPtr frm,
                     Word labelID,
                     CharPtr newLabel)
```

Parameters		
	frm	Pointer to memory block that contains the form.
	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. Redraw the label if the form's `usable` attribute and the label's `visible` attribute are set.

See Also: `FrmGetLabel`

FrmCopyTitle

Purpose	Copy the title passed over the form's current title. If the form is visible, the new title is drawn.
Prototype	`void FrmCopyTitle (FormPtr frm, CharPtr newTitle)`
Parameters	frm — Memory block that contains the form. 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 string to be assigned dynamically.
See Also	`FrmGetTitle`

FrmCustomAlert

Purpose	Create a modal dialog from an alert resource and display the dialog until the user taps a button in the alert dialog.
Prototype	`Word FrmCustomAlert ( Word alertId, CharPtr s1, CharPtr s2, CharPtr s3)`
Parameters	alertId — Resource ID of the alert. s1, s2, s3 — Strings to replace ^1, ^2, and ^3.
Result	Returns the button number the user tapped (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 "text replacement variables" ^1, ^2 and ^3 that are contained in the message string of the alert resource.
See Also	`FrmAlert`, `FrmDoDialog`

FrmDeleteForm

Purpose	Release the memory occupied by a form. Any memory allocated to objects in the form is also released.
Prototype	`void FrmDeleteForm (FormPtr frm)`
Parameters	frm Pointer to memory block that contains the form.
Result	Returns nothing.
Comments	This function does not modify the display.
See Also	`FrmInitForm`, `FrmReturnToForm`

FrmDispatchEvent

Purpose	Dispatch an event to the application's handler for the form.
Prototype	`Boolean FrmDispatchEvent (EventPtr eventP)`
Parameters	eventP Pointer to an event.
Result	Returns nothing.
Comments	The event is dispatched to the current form unless the form ID is specified in the event data, as, for example, with `frmOpenEvent`.
See Also	`FrmSetEventHandler`, `FrmHandleEvent`

FrmDoDialog

Purpose	Display a modal dialog until the user taps a button in the dialog.
Prototype	`Word FrmDoDialog (FormPtr frm)`
Parameters	frm Pointer to memory block that contains the form.
Result	Returns the number of the button the user tapped (first button is zero).
Comments	A button's item number is determined by its order in the alert template; the first button has an item number of 0 (zero).
See Also	`FrmInitForm`, `FrmCustomAlert`

FrmDrawForm

Purpose	Draw all objects in a form and the frame around the form.
Prototype	`void FrmDrawForm (FormPtr frm)`
Parameters	frm Pointer to the memory block that contains the form.
Result	Returns nothing.
Comments	Saves the bits behind the form using the `bitsBehindForm` field.
See Also	`FrmEraseForm`, `FrmInitForm`

FrmEraseForm

Purpose	Erase a form from the display.
Prototype	`void FrmEraseForm (FormPtr frm)`
Parameters	frm Pointer to the memory block that contains the form.
Result	Returns nothing.
Comments	If the region obscured by the form was saved by `FrmDrawForm`, this function restores that region.
See Also	`FrmDrawForm`

FrmGetActiveForm

Purpose	Return the currently active form.
Prototype	`FormPtr FrmGetActiveForm (void)`
Parameters	None.
Result	Returns the pointer to the memory block that contains the form.
See Also	`FrmGetActiveFormID`, `FrmSetActiveForm`

FrmGetActiveFormID

Purpose	Return the ID of the currently active form.
Prototype	`Word FrmGetActiveFormID (void)`
Parameters	None.
Result	Returns the currently active form's ID number.
See Also	`FrmGetActiveForm`

FrmGetControlGroupSelection

Purpose	Return the item number of the control selected in a group of controls.

Prototype

```
Byte FrmGetControlGroupSelection ( FormPtr frm,
                                   Byte groupNum)
```

Parameters		
	frm	Pointer to memory block that contains the form.
	groupNum	Control group number.

Result: Returns the item number of the selected control, -1 if none is selected.

Comments: The item number is the index into the form's `objects` data structure.

See Also: `FrmGetObjectId`, `FrmGetObjectPtr`, `FrmSetControlGroupSelection`

FrmGetControlValue

Purpose: Return the on/off state of a control.

Prototype

```
short FrmGetControlValue ( FormPtr frm,
                           Word objIndex)
```

Parameters		
	frm	Pointer to memory block that contains the form.
	objIndex	Item number of the object.

Result: Returns the state of the control: 1 = on; 0 = off.

Comments: The caller must specify a valid index. This function is used only for push button and check box control objects.

See Also: `FrmGetObjectIndex`, `FrmSetControlValue`

FrmGetFirstForm

Purpose — Return the first form in the window list.

Prototype — `FormPtr FrmGetFirstForm (void)`

Parameters — None.

Result — Returns a pointer to a form, 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 (UI element) that has the focus.

Prototype — `Word FrmGetFocus (FormPtr frm)`

Parameters

frm	Pointer to memory block that contains the form.

Result — Returns the index of the object (UI element) that has the focus, or -1 if none does.

See Also — `FrmGetObjectId`, `FrmGetObjectPtr`, `FrmSetFocus`

FrmGetFormBounds

Purpose — Return the visual bounds of the form; the region returned includes the form's frame.

Prototype — `void FrmGetFormBounds (FormPtr frm, RectanglePtr r)`

Parameters

frm	Pointer to memory block that contains the form.
r	Pointer to a `RectangleType` structure that will contain the bounds.

Result — Returns bounds of the form in `r`.

FrmGetFormId

Purpose	Return the resource ID of a form.
Prototype	`Word FrmGetFormId (FormPtr frm)`
Parameters	frm — Pointer to memory block that contains the form.
Result	Returns form resource ID.
See Also	`FrmGetFormPtr`

FrmGetFormPtr

Purpose	Return a pointer to the form that has the specified ID.
Prototype	`FormPtr FrmGetFormPtr (Word formId)`
Parameters	formId — Form ID number.
Result	Returns a pointer to the memory block that contains the form, 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.

Prototype

```
VoidPtr FrmGetGadgetData ( FormPtr frm,
                           Word objIndex)
```

Parameters

frm	Pointer to memory block that contains the form.
objIndex	Item number of the gadget object.

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`

FrmGetLabel

Purpose	Return pointer to the text of the specified label object in the specified form.

Prototype

```
CharPtr FrmGetLabel (FormPtr frm, Word labelID)
```

Parameters

frm	Pointer to memory block that contains the form.
labelID	ID of the label object.

Result	Returns 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.	
Prototype	`Word FrmGetNumberOfObjects (FormPtr frm)`	
Parameters	frmPtr	Pointer to memory block that contains the form.
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.

Prototype
```
void FrmGetObjectBounds (  FormPtr frm,
                           Word ObjIndex,
                           RectanglePtr r)
```

Parameters	frm	Pointer to memory block that contains the form.
	ObjIndex	Index of an object in the form.
	r	Pointer to the rectangle containing the object bounds.

Result Returns nothing. The object's bounds are returned in `r`.

See Also `FrmGetObjectPositon`, `FrmGetObjectIndex`, `FrmSetObjectPositon`

FrmGetObjectId

Purpose	Return the ID of the specified object.
Prototype	`Word FrmGetObjectId (FormPtr frm, Word objIndex)`
Parameters	frm — Pointer to memory block that contains the form. objIndex — Index of an object in the form.
Result	Returns the ID number of a object.
Comments	The application developer specifies a unique object ID.
See Also	`FrmGetObjectPtr`, `FrmGetObjectIndex`

FrmGetObjectIndex

Purpose	Return the item number of an object. The item number is the position of the object in the form's objects list.
Prototype	`Word FrmGetObjectIndex (FormPtr frm, Word objID)`
Parameters	frmPtr — Pointer to memory block that contains the form. objID — ID of an object in the form.
Result	Returns the item number of an object (the first item number is 0).
See Also	`FrmGetObjectPtr`, `FrmGetObjectId`

FrmGetObjectPositon

Purpose	Return the coordinate of the specified object relative to the form.

Prototype

```
void FrmGetObjectPositon ( FormPtr frm,
                           Word objIndex,
                           SWordPtr x, SWordPtr y)
```

Parameters

frm	Pointer to memory block that contains the form.
objIndex	Item number of the object.
x	Pointer to window-relative x position.
y	Pointer to window-relative y position.

Result	Returns nothing.
Comment	The function name is misspelled (the second “i” is missing).
See Also	`FrmGetObjectBounds`, `FrmSetObjectPositon`

FrmGetObjectPtr

Purpose	Return a pointer to the data structure of an object in a form.

Prototype

```
void * FrmGetObjectPtr (FormPtr frm, Word objIndex)
```

Parameters

frm	Pointer to memory block that contains the form.
objIndex	Item number of the object.

Result	Returns pointer to an object in the form.
See Also	`FrmGetObjectIndex`, `FrmGetObjectId`

FrmGetObjectType

Purpose	Return the type of an object.
Prototype	`FormObjectKind FrmGetObjectType ( FormPtr frm, Word objIndex)`
Parameters	frm — Pointer to memory block that contains the form. objIndex — Item number of the object.
Result	Returns `FormObjectKind` of the item specified.

FrmGetTitle

Purpose	Return a pointer to the title string of a form.
Prototype	`CharPtr FrmGetTitle (FormPtr frm)`
Parameters	frm — Pointer to memory block that contains the form.
Result	Returns a pointer to title string.
Comments	This is a pointer to the internal structure, **not** a copy.
See Also	`FrmCopyTitle`, `FrmSetTitle`

FrmGetUserModifiedState

Purpose	Return TRUE if an object in the form has been modified by the user since it was initialized or since last call to `FrmSetNotUserModified`.
Prototype	`Boolean FrmGetUserModifiedState (FormPtr frm)`
Parameters	frm Pointer to the memory block that contains the form.
Result	Returns TRUE if an object was modified, FALSE otherwise.
Comments	Returns TRUE if the `dirty` attribute of the form has been set.
See Also	`FrmSetNotUserModified`

FrmGetWindowHandle

Purpose	Return the window handle of a form.
Prototype	`WinHandle FrmGetWindowHandle (FormPtr frm)`
Parameters	frm Pointer to memory block that contains the form.
Result	Returns the handle of the memory block that the form is in. Since the form structure begins with the `WindowType` structure, this is also a `WinHandle`.

FrmGotoForm

Purpose
Send a `frmCloseEvent` to the current form; send a `frmLoadEvent` and a `frmOpenEvent` to the specified form.

Prototype
```
void FrmGotoForm (Word formId)
```

Parameters

formId	ID of the form to display.

Result
Returns nothing.

Comments
The 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.

Prototype
```
Boolean FrmHandleEvent ( FormPtr frm,
                         EventPtr event)
```

Parameters

frm	Pointer to the memory block that contains the form.
event	Pointer to the event data structure.

Result
Returns TRUE if the event was handled.

See Also
`FrmDispatchEvent`

FrmHelp

Purpose Display the specified help message until the user taps the done button in the help dialog.

Prototype
```
void FrmHelp (Word helpMsgId)
```

Parameters

helpMsgId	Resource ID of help message string.

Result Returns nothing.

Comments The ID passed is the resource ID of a string resource that contains the help message. The help message is displayed in a modal dialog with vertical scrolls if necessary.

FrmHideObject

Purpose Erase the specified object and set its attribute data so that it does not redraw or respond to the pen.

Prototype
```
void FrmHideObject (FormPtr frm, Word objIndex)
```

Parameters

frm	Pointer to memory block that contains the form.
objIndex	Item number of the object.

Result Returns nothing.

See Also `FrmGetObjectIndex`, `FrmShowObject`

FrmInitForm

Purpose	Load and initialize a form resource.
Prototype	`FormPtr FrmInitForm (Word rscID)`
Parameters	rscID — Resource ID of the form.
Result	Returns a pointer to the form memory block. Displays an error message if the form has already been initialized.
Comments	This function does not affect the display nor make the form active.
See Also	`FrmDoDialog`, `FrmDeleteForm`

FrmPopupForm

Purpose	Send a `frmOpenEvent` to the specified form. This routine differs from `FrmGotoForm` in that the current form is not closed.
Prototype	`void FrmPopupForm (Word formId)`
Parameters	formID — Resource ID of form to open.
Result	Returns nothing.
See Also	`FrmGotoForm`

FrmReturnToForm

Purpose	Erase and delete the currently active form and make the specified form the active form.
Prototype	`void FrmReturnToForm (Word 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.
See Also	`FrmGotoForm`, `FrmPopupForm`

FrmSaveAllForms

Purpose	Send a `frmSaveEvent` to all open forms.
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.
Prototype	`void FrmSetActiveForm (FormPtr frm)`
Parameters	frm Pointer to memory block that contains the form.
Result	Returns nothing.
Comments	A `penDownEvent` outside the form but within the display area is ignored.
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.

Prototype

```
void FrmSetCategoryLabel ( FormPtr frm,
                           Word objIndex,
                           CharPtr newLabel)
```

Parameters

frm	Pointer to memory block that contains the form.
objIndex	Item number of the object.
newLabel	Pointer to the name of the new category.

Result

Returns nothing.

Comments

The pointer to the new label is saved in the object.

FrmSetControlGroupSelection

Purpose	Set the selected control in a group of controls.

Prototype

```
void FrmSetControlGroupSelection (FormPtr frm,
                                  Byte groupNum,
                                  Word controlID)
```

Parameters		
	frm	Pointer to memory block that contains the form.
	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.
See Also	`FrmGetControlGroupSelection`

FrmSetControlValue

Purpose	Turn a control on or off.

Prototype

```
void FrmSetControlValue (  FormPtr frm,
                           Word objIndex,
                           short newValue)
```

Parameters		
	frm	Pointer to memory block that contains the form.
	objIndex	Item number of the object.
	newValue	New control value (non-zero equals on).

Result	Returns nothing.
Comments	The display is not changed.
See Also	`FrmGetControlValue`

FrmSetEventHandler

Purpose	Set the event handler callback routine for the specified form.

Prototype

```
void FrmSetEventHandler (FormPtr frm,
                         FormEventHandlerPtr handler)
```

Parameters

frm	Pointer to memory block that contains the form.
handler	Address of a function.

Result Returns nothing.

Comments `FrmHandleEvent` calls this handler whenever it receives an event.

This routine should be called right after a form resource is loaded. The callback routine is the mechanism for dispatching events to an application. The tutorial explains how to use callback routines.

See Also `FrmDispatchEvent`

FrmSetFocus

Purpose Set the focus of a form to the specified object.

Prototype

```
void FrmSetFocus (FormPtr frm, Word objIndex)
```

Parameters

frm	Pointer to memory block that contains the form.
objIndex	Item number of the object (UI element) that gets the focus.

Result Returns nothing.

See Also `FrmGetFocus`, `FrmGetObjectIndex`

FrmSetGadgetData

Purpose	Store the value passed in the `data` field of the gadget object.

Prototype

```
void FrmSetGadgetData (  FormPtr frm,
                         Word objIndex,
                         VoidPtr data)
```

Parameters

frmPtr	Pointer to memory block that contains the form.
objIndex	Item number of the object.
data	Application-defined value.

Result Returns nothing.

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 `FrmGetGadgetData`, `FrmGetObjectIndex`

FrmSetNotUserModified

Purpose Clear the flag that keeps track of whether or not the form has been modified by the user.

Prototype

```
void FrmSetNotUserModified (FormPtr frm)
```

Parameters

frm	Pointer to memory block that contains the form.

Result Returns nothing.

See Also `FrmGetUserModifiedState`

FrmSetObjectPositon

Purpose Set the window-relative coordinate of the specified object.

Prototype
```
void FrmSetObjectPositon ( FormPtr frm,
                           Word objIndex,
                           SWord x, SWord y)
```

Parameters

frm	Pointer to memory block that contains the form.
objIndex	Item number of the object.
x	Window-relative coordinate.
y	Window-relative coordinate.

Result Returns nothing.

Comments Does not update the display. Presently only label objects are affected.

See Also `FrmGetObjectPositon`, `FrmGetObjectIndex`, `FrmGetObjectBounds`

FrmSetTitle

Purpose Set the title of a form. If the form is visible, draw the new title.

Prototype
```
void FrmSetTitle (FormPtr frm, CharPtr newTitle)
```

Parameters

frm	Pointer to memory block that contains the form.
newTitle	Pointer to the new title string.

Result Returns nothing.

Comments Draws the title if the form is visible.

Saves the pointer to the passed title string. Does **not** make a copy.

See Also `FrmGetTitle`, `FrmCopyTitle`, `FrmCopyLabel`

FrmShowObject

Purpose	Set an object (UI element) as usable. If the form is visible, draw the object.
Prototype	`void FrmShowObject (FormPtr frm, Word objIndex)`

Parameters

frm	Pointer to memory block that contains the form.
objIndex	Item number of the object.

Result: Returns nothing.

See Also: `FrmHideObject`, `FrmGetObjectIndex`

FrmUpdateScrollers

Purpose: Visually update the field scroll arrow buttons.

Prototype

```
void FrmUpdateScrollers (  FormPtr frm,
                           Word upIndex,
                           Word downIndex,
                           Boolean scrollableUp,
                           Boolean scrollableDown)
```

Parameters

frm	Pointer to a form.
upIndex	Index of the up-scroller button.
downIndex	Index of the down-scroller button.
scrollableUp	TRUE if the up-scroll should be active.
scrollableDown	TRUE if the down-scroll should be active.

Result: Returns nothing.

See Also: `FrmGetObjectIndex`

FrmUpdateForm

Purpose	Send a `frmUpdateEvent` to the specified form.
Prototype	`void FrmUpdateForm (Word formId, Word updateCode)`
Parameters	formID — Resource ID of form to open. updateCode — If the update code is `frmRedrawUpdateCode`, the form reinitializes its global variables and redraws itself. Otherwise, the form reinitializes its global variables but does not redraw itself.
Result	Returns nothing.

FrmVisible

Purpose	Return TRUE if the form is visible (is drawn).
Prototype	`Boolean FrmVisible (FormPtr frm)`
Parameters	frm — Pointer to memory block that contains the form.
Result	Returns TRUE if visible, FALSE if not visible.
See Also	`FrmDrawForm`, `FrmEraseForm`

List UI Functions

LstDrawList

Purpose	Draw the list object if it is usable.
Prototype	`void LstDrawList (ListPtr list)`
Parameters	list — Pointer to list object (`ListType` data structure).
Result	Returns nothing.
Comments	If there are more choices than can be displayed, this function ensures that the current selection is visible. If possible, the current selection is displayed at the top. The current selection is highlighted. 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. This function sets the `visible` attribute to TRUE.
See Also	`FrmGetObjectPtr`, `LstPopupList`, `LstEraseList`

LstEraseList

Purpose	Erase a list object.
Prototype	`void LstEraseList (ListPtr ListP)`
Parameters	ListP — Pointer to a list object (`ListType` data structure).
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.
Prototype	`Word LstGetNumberOfItems (ListPtr ListP)`
Parameters	ListP — Pointer to a list object (`ListType` data structure).
Result	Returns the number of items in a list.
See Also	`FrmGetObjectPtr`, `LstSetListChoices`

LstGetSelection

Purpose	Return the currently selected choice in the list. If there is no selection, return `NoListSelection` (-1).
Prototype	`Word LstGetSelection (ListPtr ListP)`
Parameters	ListP — Pointer to list object.
Result	Returns the item number of the current list choice. The list choices are numbered sequentially, starting with 0; -1 = none.
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.

Prototype

```
CharPtr LstGetSelectionText (  ListPtr ListP,
                               Word itemNum)
```

Parameters		
	ListP	Pointer to list object.
	itemNum	Item to select (0 = first item in list).

Result	Returns pointer to the text of the current selection, or NULL if out of bounds.
Comments	This is a pointer within `ListType` structure, not a copy.
See Also	`FrmGetObjectPtr`, `LstSetListChoices`

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).

Prototype

```
Boolean pascal LstHandleEvent (ListPtr listP,
                               EventPtr pEvent)
```

Parameters

listP	Pointer to a list object (`ListType` data structure).
pEvent	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, no changes are made.
Prototype	`LstMakeItemVisible ( ListPtr ListP, Word itemNum)`
Parameters	ListP — Pointer to a list object (`ListType` data structure). 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`

LstPopupList

Purpose	Display a modal window that contains the items in the list.
Prototype	`short LstPopupList (ListPtr ListP)`
Parameters	ListP — Pointer to 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`

LstSetDrawFunction

Purpose: Set a callback function to draw each item instead of drawing the item's text string.

Prototype:
```
void LstSetDrawFunction (ListPtr list,
                         ListDrawDataFuncPtr func)
```

Parameters:

list	Pointer to list object.
func	Pointer to function which 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`

LstSetHeight

Purpose: Set the number of items visible in a list.

Prototype:
```
void LstSetHeight (  ListPtr ListP,
                     Word visibleItems)
```

Parameters:

ListP	Pointer to list object.
visibleItems	Number of choices visible at once.

Result: Returns nothing.

Comments: This function does not redraw the list if it is already visible.

See Also: `FrmGetObjectPtr`

LstSetListChoices

Purpose Set the items of a list to the array of text strings passed to this function. This function does not affect the display of the list.

Prototype
```
void LstSetListChoices ( ListPtr ListP,
                         char ** itemsText,
                         UInt numItems)
```

Parameters

ListP	Pointer to a list object.
itemsText	Pointer to an array of text strings.
numItems	Number of choices in the list.

Result Returns nothing.

Comments If the list is visible, erases the old list items.

See Also `FrmGetObjectPtr`, `LstSetSelection`, `LstSetTopItem`, `LstDrawList`, `LstSetHeight`, `LstSetDrawFunction`

LstSetPosition

Purpose Set the position of a list.

Prototype
```
void LstSetPosition (  ListPtr ListP,
                       short x,
                       short y)
```

Parameters

ListP	Pointer to a list object
x	Left bound.
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.
Prototype	`void LstSetSelection ( ListPtr ListP, Word itemNum)`
Parameters	ListP — Pointer to a list object. itemNum — Item to select (0 = first item in list, -1 = 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.
See Also	`FrmGetObjectPtr`, `LstSetSelection`

LstSetTopItem

Purpose	Set the item visible. The item cannot become the top item if it's on the last page.
Prototype	`void LstSetTopItem (ListPtr ListP, UInt itemNum)`
Parameters	ListP — Pointer to list object. itemNum — Item to select (0 = first item in list).
Result	Returns nothing.
Comments	Does **not** update the display.
See Also	`FrmGetObjectPtr`, `LstSetSelection`, `LstMakeItemVisible`, `LstDrawList`, `LstEraseList`

Menu Functions

MenuDispose

Purpose	Release any memory allocated to support the menu management.
Prototype	`void MenuDispose (MenuBarPtr MenuP)`
Parameters	MenuP — Pointer returned by `MenuInit`; this is a pointer to a `MenuBarType` data structure.
Result	Returns nothing.
Comments	This function is useful for applications that have multiple menu bars. It frees all memory allocated by a menu, resets the command status, and restores the saved bits to the screen.
See Also	`MenuInit`, `MenuDrawMenu`

MenuDrawMenu

Purpose	Draw the current menu bar and the last pull-down that was visible.
Prototype	`void MenuDrawMenu (MenuBarPtr MenuP)`
Parameters	MenuP — Pointer to a `MenuBarType` data structure.
Result	Returns nothing.
Comments	If a pull-down menu was visible the last time the menu bar was visible, the pull-down menu is also drawn. The first time a menu bar is drawn no pull-down menu is displayed. The menu bar and the pull-down menu are drawn in front of all the applications windows. Screen regions obscured by the menus are saved by this function and restored by `MenuEraseStatus`.
See Also	`MenuInit`, `MenuEraseStatus`, `MenuDispose`

MenuEraseStatus

Purpose	Erase the menu command status.
Prototype	`void MenuEraseStatus (MenuBarPtr MenuP)`
Parameters	MenuP — Pointer to a `MenuBarType` data structure, or NULL for the current menu.
Result	Returns nothing.
See Also	`MenuInit`

MenuGetActiveMenu

Purpose	Returns a pointer to the current menu.
Prototype	`MenuBarPtr MenuGetActiveMenu (void)`
Parameters	None.
Result	Returns a pointer to the current menu, NULL if there is none.
See Also	`MenuSetActiveMenu`

MenuHandleEvent

Purpose
Handle events in the current menu. This routine handles two types of events, `penDownEvent` and `winEnterEvent`.

Prototype
```
Boolean MenuHandleEvent (  MenuBarPtr MenuP,
                           EventPtr event,
                           WordPtr error)
```

Parameters

MenuP	Pointer to a `MenuBarType` data structure.
event	Pointer to an `EventType` structure.
error	Error (or 0 if no error).

Result
Returns TRUE if the event is handled. (If the event is a `penDownEvent` within the menu bar or the menu, or the event is a `keyDownEvent` that the menu supports.)

Comments
When `MenuHandleEvent` receives a `penDownEvent`, it checks if the pen position is within the bounds of the menu object. If it is, `MenuHandleEvent` tracks the pen until it comes up. If the pen comes up within the bounds of the menu, a `winEnterEvent` is added to the event queue, and the routine is exited.

When `MenuHandleEvent` receives a `winEnterEvent`, it checks that the menu ID in the event record matches the ID of the specified menu. If there is a match, `MenuHandleEvent` creates and displays a popup window containing the menu's choices, and the routine is exited.

If a `penDownEvent` is received while the menu's popup window is displayed, and the pen position is outside the bounds of the popup window, the menu is dismissed. If the pen position is within the bounds of the window `MenuHandleEvent` tracks the pen until it comes up. If the pen comes up in the menu, a `winExitEvent` is added to the event queue.

MenuInit

Purpose	Load a menu resource from a resource file.
Prototype	`MenuBarPtr MenuInit (Word 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` data structure).
Comments	The menu is not usable until `MenuSetActiveMenu` is called.
See Also	`MenuSetActiveMenu`, `MenuDispose`

MenuSetActiveMenu

Purpose	Set the current menu.
Prototype	`MenuBarPtr MenuSetActiveMenu (MenuBarPtr 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.
See Also	`MenuGetActiveMenu`

9

Table Functions

TblDrawTable

Purpose	Draw a table.
Prototype	`void TblDrawTable (TablePtr table)`
Parameters	table — Pointer to a table object.
Result	Returns nothing.
See Also	`TblEraseTable`, `TblRedrawTable`, `TblSetCustomDrawProcedure`

TblEditing

Purpose	Check whether a table is in edit mode.
Prototype	`Boolean TblEditing (TablePtr table)`
Parameters	table — Pointer to a table object.
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.

TblEraseTable

Purpose	Erase a table object.
Prototype	`void TblEraseTable (TablePtr table)`
Parameters	table — Pointer to a table object.
Result	Returns nothing.
See Also	`TblDrawTable`, `TblSetCustomDrawProcedure`, `TblRedrawTable`

TblFindRowData

Purpose: Return the row number that contains the specified data value.

Prototype:

```
Boolean TblFindRowData ( TablePtr table,
                         ULong data,
                         WordPtr rowP)
```

Parameters:

table	Pointer to a table object.
data	Row data to find.
rowP	Pointer to the row number (return value).

Result: Returns TRUE if a match was found, FALSE otherwise.

See Also: `TblGetRowData`, `TblFindRowID`

TblFindRowID

Purpose	Return the number of the row that matches the specified ID.

Prototype

```
Boolean TblFindRowID ( TablePtr table,
                       Word id,
                       WordPtr rowP)
```

Parameters

table	Pointer to a table object.
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: `TblFindRowData`

TblGetBounds

Purpose: Return the bounds of a table.

Prototype

```
void TblGetBounds (TablePtr table, RectanglePtr r)
```

Parameters

table	Pointer to a table object.
r	Pointer to a `RectangleType` structure.

Result: Returns nothing. Stores the bounds in `r`.

See Also: `TblGetItemBounds`

TblGetColumnSpacing

Purpose Return the spacing after the specified column.

Prototype
```
Word TblGetColumnSpacing ( TablePtr table,
                           Word column)
```

Parameters

table	Pointer to a table object.
column	Column number (zero-based).

Result Returns the spacing after column (in pixels).

See Also `TblGetColumnWidth`, `TblSetColumnSpacing`, `TblSetColumnUsable`

TblGetColumnWidth

Purpose Return the width of the specified column.

Prototype
```
Word TblGetColumnWidth ( TablePtr table,
                         Word column)
```

Parameters

table	Pointer to a table object.
column	Column number (zero-based).

Result Returns the width of a column (in pixels).

See Also `TblGetColumnSpacing`, `TblSetColumnWidth`, `TblSetColumnUsable`

TblGetCurrentField

Purpose Return a pointer to the field structure in which the user is currently editing a text item.

Prototype `FieldPtr TblGetCurrentField (TablePtr table)`

Parameters

table	Pointer to a table object.

Result Returns `FieldPtr`, or NULL if the table is not in edit mode.

See Also `TblGetSelection`

TblGetItemBounds

Purpose Return the bounds of an item in a table.

Prototype

```
void TblGetItemBounds (  TablePtr table,
                         Word row, Word column,
                         RectanglePtr r)
```

Parameters

table	Pointer to a table object.
row	Row of the item (zero-based).
column	Column of the item (zero-based).
r	Pointer to a structure that holds the bounds of the item.

Result Returns nothing. Stores the bounds in `r`.

TblGetItemInt

Purpose	Return the integer value stored in a table item.
Prototype	`Word TblGetItemInt ( TablePtr table, Word row, Word column)`

```
Word TblGetItemInt (   TablePtr table,
                       Word row, Word column)
```

Parameters

table	Pointer to a table object.
row	Row of the item to select (zero-based).
column	Column of the item to select (zero-based).

Result — Returns the integer value.

See Also — `TblSetItemInt`

TblGetLastUsableRow

Purpose — Return the last row in a table that is usable (visible).

Prototype

```
Word TblGetLastUsableRow (TablePtr table)
```

Parameters

table	Pointer to a table object.

Result — Returns the row index (zero-based) or -1 if there are no usable rows.

See Also — `TblGetRowData`, `TblGetRowID`

TblGetNumberOfRows

Purpose — Return the number of rows in a table.

Prototype

```
Word TblGetNumberOfRows (TablePtr table)
```

Parameters

table	Pointer to a table object.

Result — Returns the number of rows in the specified table.

TblGetRowData

Purpose	Return the data value of the specified row.
Prototype	`ULong TblGetRowData (TablePtr table, Word row)`
Parameters	table — Pointer to a table object.
	row — Row of the item to select (zero-based).
Comments	The data value is a placeholder for application-specific values.
See Also	`TblGetRowID`, `TblSetRowData`

TblGetRowHeight

Purpose	Return the height of the specified row.
Prototype	`Word TblGetRowHeight (TablePtr table, Word row)`
Parameters	table — Pointer to a table object.
	row — Row to get (zero-based).
Result	Returns the height in pixels.
See Also	`TblGetItemBounds`, `TblSetRowHeight`

TblGetRowID

Purpose	Return the ID value of the specified row.
Prototype	`Word TblGetRowID (TablePtr table, Word row)`
Parameters	table — Pointer to a table object.
	row — Row for which the ID will be returned (zero-based).
Result	Returns the ID value of the row in the table.
See Also	`TblGetRowData`, `TblSetRowHeight`

TblGetSelection

Purpose	Return the row and column of the currently selected table item.

Prototype

```
Boolean TblGetSelection (  TablePtr table,
                           WordPtr rowP,
                           WordPtr columnP)
```

Parameters

table	Pointer to a table object.
rowP	Pointer to a Word variable in which to store the row (zero-based).
columnP	Pointer to a Word variable in which to store the column (zero-based).

Result Returns TRUE if the item is highlighted, FALSE if not.

See Also `TblSetRowSelectable`

TblGrabFocus

Purpose Put a table into edit mode.

Prototype

```
void TblGrabFocus (  TablePtr table,
                     Word row, Word column)
```

Parameters

table	Pointer to a table object.
row	Current row to be edited (zero-based).
column	Current column to be edited (zero-based).

Result Returns nothing.

Comments Displays an error if the row or column passed is out of bounds. An editable field must exist in the coordinates passed to this function.

See Also `TblReleaseFocus`

TblHandleEvent

Purpose	Handle an event for the table.
Prototype	`Boolean TblHandleEvent ( TablePtr table, EventPtr event)`
Parameters	table — Pointer to a table object. event — The event to be handled.
Result	Returns TRUE if the event was handled, FALSE if it was not.

TblInsertRow

Purpose	Insert a row into the table before the specified row. The number of rows in the table is not increased; the last row in the table is removed.
Prototype	`void TblInsertRow (TablePtr table, Word row)`
Parameters	table — Pointer to a table object. row — Row to insert (zero-based).
Result	Returns nothing.
Comments	If the `row` parameter is greater than or equal to the number of rows in the table, an error is displayed.
See Also	`TblRemoveRow`, `TblSetRowUsable`, `TblSetRowSelectable`, `TblMarkRowInvalid`

TblMarkRowInvalid

Purpose	Mark the image of the specified row invalid.
Prototype	`void TblMarkRowInvalid (TablePtr table, Word row)`
Parameters	table — Pointer to a table object. row — Row of the item to select (zero-based).
Comments	After calling this function, call `TblRedrawTable` to redraw all rows marked invalid. Rows not marked invalid are not redrawn.
Result	Returns nothing.
See Also	`TblRemoveRow`, `TblSetRowUsable`, `TblSetRowSelectable`, `TblMarkTableInvalid`, `TblRowInvalid`

TblMarkTableInvalid

Purpose	Mark the image of all the rows in a table invalid.
Prototype	`void TblMarkTableInvalid (TablePtr table)`
Parameters	table — Pointer to a table object.
Result	Returns nothing.
Comments	After calling this function, you must call `TblRedrawTable` to redraw all rows. Rows not marked invalid do not draw.
See Also	`TblEraseTable`, `TblRedrawTable`, `TblMarkTableInvalid`

TblRedrawTable

Purpose	Redraw the rows of the table that are marked invalid.
Prototype	`void TblRedrawTable (TablePtr table)`
Parameters	table — Pointer to a table object.
Result	Returns nothing.
See Also	`TblMarkTableInvalid`

TblReleaseFocus

Purpose	Release the focus.
Prototype	`void TblReleaseFocus (TablePtr table)`
Parameters	table — Pointer to a table object.
Result	Returns nothing.
Comments	If the current item is a text item, the memory allocated for editing is released and the insertion point is turned off.
See Also	`TblGrabFocus`

TblRemoveRow

Purpose	Remove the specified row from the table.
Prototype	`void TblRemoveRow (TablePtr table, Word row)`
Parameters	table — Pointer to a table object. row — Row to remove (zero-based).
Result	Returns nothing.
Comments	The number of rows in the table is not decreased; an unusable row is added to the end of the table. If an invalid row is specified, an error is displayed. This function does not visually update the display.
See Also	`TblInsertRow`, `TblSetRowUsable`, `TblSetRowSelectable`, `TblMarkRowInvalid`

TblRowInvalid

Purpose	Determine whether a row is invalid. Invalid rows need to be re-drawn.
Prototype	`Boolean TblRowInvalid (TablePtr table, Word row)`
Parameters	table — Pointer to a table object. row — Row number (zero-based).
Result	Returns TRUE if the row is invalid, FALSE if it's valid.
See Also	`TblMarkRowInvalid`

TblRowSelectable

Purpose	Determine whether the specified row is selectable. Rows that are not selectable don't highlight when touched.
Prototype	`Boolean TblRowSelectable (TablePtr table, Word row)`
Parameters	table — Pointer to a table object.
	row — Row of the item to select (zero-based).
Result	Returns TRUE if the row is selectable, FALSE if it's not.

TblRowUsable

Purpose	Determine whether the specified row is usable.
Prototype	`Boolean TblRowUsable (TablePtr table, Word row)`
Parameters	table — Pointer to a table object.
	row — Row number (zero-based).
Result	Returns TRUE if the row is usable, FALSE if it's not.
Comments	Rows that are not usable do not display.
See Also	`TblRowSelectable`, `TblGetLastUsableRow`

TblSelectItem

Purpose Select (highlight) the specified item. If there is already a selected item, it is unhighlighted.

Prototype
```
void TblSelectItem ( TablePtr table,
                     Word row,
                     Word column)
```

Parameters

table	Pointer to a table object.
row	Row of the item to select (zero-based).
column	Column of the item to select (zero-based).

Result Returns nothing.

See Also `TblRowSelectable`, `TblGetItemBounds`, `TblGetItemInt`

TblSetColumnSpacing

Purpose Set the spacing after the specified `column`.

Prototype
```
void TblSetColumnSpacing ( TablePtr table,
                           Word column,
                           Word spacing)
```

Parameters

table	Pointer to a table object.
column	Column number (zero-based).
spacing	Spacing after the column.

Result Returns nothing.

See Also `TblSetColumnUsable`

TblSetColumnUsable

Purpose Set a column in a table usable or unusable.

Prototype
```
void TblSetColumnUsable (  TablePtr table,
                           Word row,
                           Boolean usable)
```

Parameters

table	Pointer to a table object.
column	Column of the item to select (zero-based).
usable	True for usable or false for not usable.

Result Returns nothing.

Comments Columns that are not usable do not display.

See Also `TblMarkRowInvalid`

TblSetColumnWidth

Purpose Set the width of the specified column.

Prototype
```
void TblSetColumnWidth ( TablePtr table,
                         Word column,
                         Word width)
```

Parameters

table	Pointer to a table object.
column	Column number (zero-based).
width	Width of the column (in pixels).

Result Returns nothing.

See Also `TblGetColumnWidth`

TblSetCustomDrawProcedure

Purpose Set the custom draw callback procedure for the column specified.

Prototype

```
void TblSetCustomDrawProcedure(TablePtr table,
                               Word column,
                               VoidPtr drawCallback)
```

Parameters

table	Pointer to a table object.
column	Column of table.
drawCallback	Callback function.

Note: The callback procedure should have this prototype:

```
void drawCallback (
             VoidPtr table,
             Word row,
             Word column,
             RectanglePtr bounds);
```

Result Returns nothing.

Comments The custom draw callback function is used to draw table items with a `TableItemStyleType` of `customTableItem` (see table.h).

See Also `TblDrawTable`

TblSetItemInt

Purpose Set the integer value of the specified item.

Prototype
```
void TblSetItemInt (   TablePtr table,
                       Word row, Word column,
                       Word value)
```

Parameters

table	Pointer to a table object.
row	Row of the item (zero-based).
column	Column of the item (zero-based).
value	Any byte value (an integer).

Result Returns nothing.

Comments An application can store what it wants in an item’s integer value.

See Also `TblGetItemInt`, `TblSetItemPtr`

TblSetItemPtr

Purpose Set the item to the specified pointer value.

Prototype
```
void TblSetItemPtr (   TablePtr table,
                       Word row, Word column,
                       VoidPtr value)
```

Parameters

table	Pointer to a table object.
row	Row of the item (zero-based).
column	Column of the item (zero-based).
value	Pointer to data to display in the table item.

Result Returns nothing.

Comments An application can store whatever it wants in the table item.

See Also `TblSetItemInt`

TblSetItemStyle

Purpose	Set the item to display its data in a style; for example, text, numbers, dates, and so on.

Prototype

```
void TblSetItemStyle ( TablePtr table,
                       Word row,
                       Word column,
                       TableItemStyleType type)
```

Parameters

table	Pointer to a table object.
row	Row of the item (zero-based).
column	Column of the item (zero-based).
type	See Table.h.

Result Returns nothing.

See Also `TblSetCustomDrawProcedure`

TblSetLoadDataProcedure

Purpose Set the load-data callback procedure for the specified column.

Prototype
```
void TblSetLoadDataProcedure(TablePtr table,
             Word column,
             TableLoadDataFuncPtr loadDataCallback)
```

Parameters

table	Pointer to a table object.
column	Column of table.
loadDataCallback	Callback procedure.

Note: The callback procedure should have this prototype:

```
VoidHand LoadDataCallback
             (VoidPtr table,
              Word row,
              Word column,
              Boolean editable,
              WordPtr dataOffset,
              WordPtr dataSize);
```

For a text style item, the callback procedure should return the handle of a block that contains a null-terminated text string, the offset from the start of the block to the start of the string, and the amount of space allocated for the string.

Result Returns nothing.

Comments The callback function is used to obtain the data values of a table item.

See Also `TblSetCustomDrawProcedure`

TblSetRowData

Purpose	Set the data value of the specified row. The data value is a placeholder for application-specific values.

Prototype
```
void TblSetRowData ( TablePtr table,
                     Word row,
                     ULong data)
```

Parameters

table	Pointer to a table object.
row	Row of the item to select (zero-based).
data	Application-specific data.

Result: Returns nothing.

See Also: `TblGetRowData`

TblSetRowHeight

Purpose: Set the height of the specified row.

Prototype
```
void TblSetRowHeight ( TablePtr table,
                       Word row, Word height)
```

Parameters

table	Pointer to a table object.
row	Row to set (zero-based).
height	New height in pixels.

Result: Returns nothing.

See Also: `TblGetRowHeight`

TblSetRowID

Purpose	Set the ID value of the specified row.
Prototype	`void TblSetRowID ( TablePtr table, Word row, Word id)`

```
void TblSetRowID ( TablePtr table,
                   Word row, Word id)
```

Parameters

table	Pointer to a table object.
row	Row of the item to select (zero-based).
id	ID to identify a row.

Result: Returns nothing.

See Also: TblGetRowID

TblSetRowSelectable

Purpose: Set a row in a table to selectable or nonselectable.

Prototype

```
void TblSetRowSelectable ( TablePtr table,
                           Word row,
                           Boolean selectable)
```

Parameters

table	Pointer to a table object.
row	Row of the item to select (zero-based).
selectable	TRUE or FALSE.

Result: Returns nothing.

Comments: Rows that are not selectable don’t highlight when touched.

See Also: TblRowSelectable, TblSetRowUsable

TblSetRowUsable

Purpose	Set a row in a table to usable or unusable. (Rows that are not usable do not display.)

Prototype

```
void TblSetRowUsable ( TablePtr table,
                       Word row,
                       Boolean usable)
```

Parameters

table	Pointer to a table object.
row	Row of the item to select (zero-based).
usable	TRUE or FALSE.

Result Returns nothing.

See Also `TblRowUsable`, `TblSetRowSelectable`

TblSetSaveDataProcedure

Purpose Set the save-data callback procedure for the specified column.

Prototype

```
void TblSetSaveDataProcedure(TablePtr table,
                             Word column,
                             VoidPtr saveDataCallback)
```

Parameters

table	Pointer to a table object.
column	Column of table.
saveDataCallback	Callback function.

Note: The callback procedure should have this prototype:

```
VoidPtr SaveDataCallback
             (VoidPtr table,
              Word row,
              Word column);
```

Comments The callback procedure is called when the table object determines the data of a text object needs to be saved.

Result Returns nothing.

See Also `TblSetCustomDrawProcedure`

TblUnhighlightSelection

Purpose Unhighlight the currently selected item in a table.

Prototype

```
void TblUnhighlightSelection (TablePtr table)
```

Parameters

table	Pointer to a table object.

Result Returns nothing.

10

Window Functions

WinAddWindow

Purpose	Add the specified window to the active windows list.
Prototype	`void WinAddWindow (WinHandle winHandle)`
Parameters	winHandle Handle of a window.
Result	Returns nothing.
Comment	The active windows list contains all windows in the current application's user interface.
See Also	`WinCreateWindow`, `WinRemoveWindow`

WinClipRectangle

Purpose	Clip a rectangle to the clipping rectangle of the draw window.
Prototype	`void WinClipRectangle (RectanglePtr r)`
Parameters	r Pointer to a structure holding the rectangle to clip.
Comment	The draw window is the window to which all drawing functions send their output. The rectangle returned in `r` is the intersection of the rectangle passed and the draw window's clipping bounds.
Result	Returns nothing.
See Also	`WinCopyRectangle`, `WinDrawRectangle`, `WinEraseRectangle`, `WinGetClip`

WinCopyRectangle

Purpose	Copy a rectangular region from one place to another (either between windows or within a single window).

Prototype

```
void WinCopyRectangle (WinHandle srcWin,
                       WinHandle dstWin,
                       RectanglePtr srcRect,
                       SWord destX,
                       SWord destY,
                       ScrOperation mode)
```

Parameters

srcWin	Window from which the rectangle is copied.
dstWin	Window to which the rectangle is copied.
srcRect	Bounds of the region to copy.
destX	Top bound of the rectangle in destination window.
destY	Left bound of the rectangle in destination window.
mode	The method of transfer from the source to the destination window (see window.h).

Result Returns nothing.

Comments Copies the bits of the window inside the rectangle region.

WinCreateWindow

Purpose
Create a new window and add it to the window list.

Prototype
```
WinHandle WinCreateWindow (  RectanglePtr bounds,
                             FrameType frame,
                             Boolean modal,
                             Boolean focusable,
                             WordPtr error)
```

Parameters

bounds	Display relative bounds of the window.
frame	Type of frame around the window (see window.h).
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 handle for the new window.

Comments
Windows created by this routine draw to the display, see `WinCreateOffscreenWindow`.

New windows are created disabled, and must be enabled before they accept input.

See Also
`WinCreateOffscreenWindow`, `WinDeleteWindow`, `WinInitializeWindow`

WinCreateOffscreenWindow

Purpose Create a new off-screen window and add it to the window list.

Prototype

```
WinHandle WinCreateOffscreenWindow (
          SWord width,
          SWord height,
          WindowFormatType format,
          WordPtr error)
```

Parameters

width	Width of the window in pixels.
height	Height of the window in pixels.
format	Either `screenFormat` or `genericFormat`.
error	Pointer to any error encountered by this function.

Result Returns the handle of the new window.

Comments Windows created with this routine draw to a memory buffer instead of the display.

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.

See Also `WinCreateWindow`, `WinAddWindow`

WinDeleteWindow

Purpose Remove a window from the window list and free the memory used by the window.

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.

Result Returns nothing.

See Also `WinCreateWindow`

WinDisableWindow

Purpose Disable a window but leave it on the active windows list (list of all windows in the system).

Prototype
```
void WinDisableWindow (WinHandle winHandle)
```

Parameters

winHandle	Handle of window to disable.

Result Returns nothing.

Comments Disabled windows ignore all pen input and cannot be made the current window or the draw window. Windows are usually disabled when they are removed from the screen.

This function does not affect the visual appearance of the window.

See Also `WinEnableWindow, WinDeleteWindow`

WinDisplayToWindowPt

Purpose	Convert a display-relative coordinate to a window-relative coordinate. The coordinate returned is relative to the display window.

Prototype

```
void WinDisplayToWindowPt (  SWordPtr extentX,
                             SWordPtr extentY)
```

Parameters

extentX	Pointer to x coordinate to convert.
extentY	Pointer to y coordinate to convert.

Result: Returns nothing.

Comments: `WinWindowToDisplayPt`

WinDrawBitmap

Purpose: Draw a bitmap at the given x and y coordinates.

Prototype

```
void WinDrawBitmap ( BitmapPtr bitmapP,
                     SWord x,
                     Sword 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.

See Also: `WinEraseRectangle`

WinDrawChars

Purpose Draw the specified characters in the draw window.

Prototype
```
void WinDrawChars ( CharPtr chars,
                    Word len,
                    SWord x,
                    SWord y)
```

Parameters

chars	Pointer to the characters to draw.
len	Number of characters to draw.
x	Left bound of first character to draw.
y	Top bound of first character to draw.

Result Returns nothing.

Comment Before calling this function, you may call `WinSetUnderlineMode` and `FntSetFont`.

See Also `WinDrawInvertedChars`, `WinEraseChars`, `WinSetUnderlineMode`

WinDrawGrayLine

Purpose Draw a line in the draw window.

Prototype
```
void WinDrawGrayLine (   SWord x1, SWord y1,
                         SWord x2, SWord y2)
```

Parameters

x1	x coordinate of the start of the line.
y1	y coordinate of the start of the line.
x2	x coordinate of the end of the line.
y2	y coordinate of the end of the line.

Result Returns nothing.

See Also `WinDrawLine`

WinDrawGrayRectangleFrame

Purpose	Draw a gray rectangular frame in the draw window.

Prototype

```
void WinDrawGrayRectangleFrame ( FrameType frame,
                                 RectanglePtr r)
```

Parameters

frame	Type of frame to draw.
r	Pointer to the rectangle to frame.

Result: Returns nothing.

Comments: 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`

WinDrawInvertedChars

Purpose: Draw the specified characters inverted (background color) in the draw window.

Prototype

```
void WinDrawInvertedChars( CharPtr chars, Word len,
                           SWord x, SWord y)
```

Parameters

chars	Pointer to the characters to draw.
len	Number of characters to draw.
x, y	Left and top bound of first character to draw.

Result: Returns nothing.

Comments: The characters are drawn in the background color and the off pixels are drawn in the foreground color.

Before calling this function, you may call `WinSetUnderlineMode` and `FntSetFont`.

See Also: `WinDrawChars`

WinDrawLine

Purpose	Draw a line in the draw window.

Prototype

```
void WinDrawLine ( short x1, short y1,
                   short x2, short y2)
```

Parameters

x1	x coordinate of the start of the line.
y1	y coordinate of the start of the line.
x2	x coordinate of the end of the line.
y2	y coordinate of the end of the line.

Result Returns nothing.

See Also `WinDrawGrayLine`, `WinEraseLine`, `WinFillLine`

WinDrawRectangle

Purpose Draw a black rectangle in the draw window; the rectangle can have square or round corners.

Prototype

```
void WinDrawRectangle (  RectanglePtr r,
                         Word cornerDiam)
```

Parameters

r	Pointer to the rectangle to draw.
cornerDiam	Diameter of rounded corners. Zero for square corners.

Result Returns nothing.

Comments The `cornerDiam` parameter specifies the diameter 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 `WinFillRectangle`, `WinEraseRectangle`

WinDrawRectangleFrame

Purpose
Draw a rectangular frame around the specified region in the draw window.

Prototype
```
void WinDrawRectangleFrame ( FrameType frame,
                             RectanglePtr r)
```

Parameters

frame	Type of frame to draw.
r	Pointer to the rectangle to frame.

Result
Returns nothing.

Comments
The frame is drawn outside the specified region.

See Also
WinEraseRectangleFrame, WinGetFramesRectangle, WinDrawGrayRectangleFrame, WinDrawWindowFrame

WinDrawWindowFrame

Purpose
Draw the frame of the current drawing window.

Prototype
```
void WinDrawWindowFrame (void)
```

Parameters
None.

Result
Returns nothing.

See Also
WinDrawRectangleFrame, WinGetDrawWindow

WinEnableWindow

Purpose	Enable a window.
Prototype	`void WinEnableWindow (WinHandle winHandle)`
Parameters	winHandle Handle of the window to enable.
Result	Returns nothing.
Comments	Enabled windows accept pen input and can be made the active window. This routine does not affect the visual appearance of the window.
See Also	WinDisableWindow, WinSetActiveWindow

WinEraseChars

Purpose Erase specified characters in the draw window.

Prototype

```
void WinEraseChars ( CharPtr chars, Word len,
                     SWord x, SWord y)
```

Parameters

chars	Pointer to the characters to erase.
len	Number of characters to erase.
x	Left bound of first character to erase.
y	Top bound of first character to erase.

Result Returns nothing.

See Also `WinDrawChars`

WinEraseLine

Purpose	Erase a line in the draw window.

Prototype

```
void WinEraseLine ( SWord x1, SWord y1,
                    SWord x2, SWord y2)
```

Parameters

x1	x coordinate of the start of the line.
y1	y coordinate of the start of the line.
x2	x coordinate of the end of the line.
y2	y coordinate of the end of the line.

Result Returns nothing.

See Also `WinDrawLine`

WinEraseRectangle

Purpose Erase a rectangle in the draw window. (The rectangle can have round or square corners; see `WinDrawRectangle`.)

Prototype

```
void WinEraseRectangle (  RectanglePtr r,
                          Word cornerDiam)
```

Parameters

r	Pointer to the rectangle to erase.
cornerDiam	Diameter of rounded corners; zero for square corners.

Result Returns nothing.

See Also `WinDrawRectangle`

WinEraseRectangleFrame

Purpose	Erase a rectangular frame in the draw window.
Prototype	`void WinEraseRectangleFrame ( FrameType frame, RectanglePtr r)`
Parameters	frame — Type of frame to erase. r — Pointer to the rectangular frame.
Result	Returns nothing.
See Also	`WinDrawRectangleFrame`

WinEraseWindow

Purpose	Erase the contents of the draw window. The frame around the draw window is not erased by this routine.
Prototype	`void WinEraseWindow (void)`
Parameters	None.
Result	Returns nothing.
See Also	`WinEnableWindow`

WinFillLine

Purpose	Fill a line in the draw window with the current pattern. You can set the current pattern with `WinSetPattern`.

Prototype

```
void WinFillLine ( SWord x1, SWord y1,
                   SWord x2, SWord y2)
```

Parameters

x1	x coordinate of the start of the line.
y1	y coordinate of the start of the line.
x2	x coordinate of the end of the line.
y2	y coordinate of the end of the line.

Result Returns nothing.

See Also `WinSetPattern`, `WinDrawLine`

WinFillRectangle

Purpose Draw a rectangle with current pattern. (The rectangle can have square or round corners.)

Prototype

```
void WinFillRectangle (  RectanglePtr r,
                         Word cornerDiam)
```

Parameters

r	Pointer to the rectangle to draw.
cornerDiam	Diameter of rounded corners. Zero for square corners.

Result Returns nothing.

Comments You can set the current pattern with `WinSetPattern`.

See Also `WinSetPattern`. `WinDrawRectangle`

WinGetActiveWindow

Purpose	Return the window handle of the active window.
Prototype	`WinHandle WinGetActiveWindow (void)`
Parameters	None.
Result	Returns the handle of the active window.
See Also	`WinSetActiveWindow`, `WinGetDisplayWindow`, `WinGetFirstWindow`, `WinGetDrawWindow`, `WinRemoveWindow`

WinGetClip

Purpose	Return the clipping rectangle of the draw window.
Prototype	`void WinGetClip (RectanglePtr r)`
Parameters	r Pointer to a structure to hold the clipping bounds.
Result	Returns nothing.
See Also	`WinSetClip`

WinGetDisplayExtent

Purpose Return the width and height of the display (the screen).

Prototype
```
void WinGetDisplayExtent ( SWordPtr extentX,
                           SWordPtr extentY)
```

Parameters

extentX	Pointer to the width of the display.
extentY	Pointer to the height of the display.

Result Returns nothing.

WinGetDisplayWindow

Purpose Return the window handle of the display window.

Prototype `WinHandle WinGetDisplayWindow (void)`

Parameters None.

Result Returns handle of display window.

Comments The display window is created by the system at start-up; its size is the same as the physical display (screen).

See Also `WinGetDisplayExtent`, `WinGetActiveWindow`, `WinGetDrawWindow`

WinGetDrawWindow

Purpose Return the window handle of the current draw window.

Prototype `WinHandle WinGetDrawWindow (void)`

Parameters None.

Result Returns handle of draw window

See Also `WinGetDisplayWindow`, `WinGetActiveWindow`, `WinSetDrawWindow`

WinGetFirstWindow

Purpose	Return a pointer to the first window in the linked list of windows.
Prototype	`WinHandle WinGetFirstWindow (void)`
Parameters	None.
Result	Returns handle of first window.
Comments	This function is usually used by the system only.
See also	`WinAddWindow`, `WinGetActiveWindow`

WinGetFramesRectangle

Purpose: Return the region needed to draw a rectangle with the specified frame around it.

Prototype:

```
void WinGetFramesRectangle (FrameType frame,
                        RectanglePtr r,
                        RectanglePtr obscuredRect)
```

Parameters:

frame	Type of frame drawn around the rectangle.
r	Pointer to the rectangle to frame.
obscuredRect	Pointer to the rectangle obscured by the frame.

Result: Returns nothing.

Comments: Frames are always drawn around (outside) a rectangle.

See Also: `WinGetWindowBounds`

WinGetPattern

Purpose	Return the current fill pattern.
Prototype	`void WinGetPattern (CustomPatternType pattern)`
Parameters	pattern — Pattern buffer to hold pattern.
Result	Returns nothing.
Comments	The fill pattern is used by `WinFillLine` and `WinFillRectangle`. A pattern defines an 8-x-8 bit pattern. The pattern is tiled to fill the specified region. The pattern structure is eight bytes long, the first byte is the first row of the pattern.
See Also	`WinSetPattern`

WinGetWindowBounds

Purpose	Return the bounds of the current draw window in display-relative coordinates.
Prototype	`void WinGetWindowBounds (RectanglePtr r)`
Parameters	r — Pointer to a rectangle.
Result	Returns nothing.
See Also	`WinGetWindowExtent`

WinGetWindowExtent

Purpose	Return the width and height of the current draw window.
Prototype	`void WinGetWindowExtent ( SWordPtr extentX, SWordPtr extentY)`
Parameters	extentX — Pointer to the width of the draw window. extentY — Pointer to the height of the draw window.
Result	Returns nothing.
See Also	`WinGetWindowBounds`, `WinGetWindowFrameRect`,

WinGetWindowFrameRect

Purpose	Return a rectangle, in display-relative coordinates, that defines the size and location of a window and its frame.
Prototype	`void WinGetWindowFrameRect ( WinHandle winHandle, RectanglePtr r)`
Parameters	winHandle — Handle of window whose coordinates are desired. r — Pointer to the coordinates of the window.
Result	Returns nothing.
See Also	`WinGetWindowBounds`

WinGetWindowPointer

Purpose	Return a pointer to the specified window's `WindowType` structure.
Prototype	`WinPtr WinGetWindowPointer (WinHandle winHandle)`
Parameters	winHandle Handle of a window.
Result	Returns nothing.
See Also	`WinGetActiveWindow`

WinInitializeWindow

Purpose	Initialize the screen-dependent members of a `WindowType` structure and set the window's clipping bounds to the window's bounds.
Prototype	`void WinInitializeWindow (WinHandle winHandle)`
Parameters	winHandle Handle of a window.
Result	Returns nothing.
See Also	`WinCreateWindow`

WinInvertChars

Purpose	Invert the specified characters in the draw window.

Prototype

```
void WinInvertChars (  CharPtr chars, Word len,
                       SWord x, SWord y)
```

Parameters

chars	Pointer to the characters to invert.
len	Number of characters to invert.
x, y	Left and top bound of first character to invert.

Result Returns nothing.

See Also `WinDrawInvertedChars`, `WinDrawChars`

WinInvertLine

Purpose Invert a line in the draw window.

Prototype

```
void WinInvertLine ( SWord x1, SWord y1,
                     SWord x2, SWord y2)
```

Parameters

x1	x coordinate of the start of the line.
y1	y coordinate of the start of the line.
x2	x coordinate of the end of the line.
y2	y coordinate of the end of the line.

Result Returns nothing.

See Also `WinInvertRectangle`, `WinInvertRectangleFrame`, `WinDrawLine`, `WinEraseLine`

WinInvertRectangle

Purpose
Invert a rectangle in the draw window. (The rectangle can have square or round corners.)

Prototype
```
void WinInvertRectangle (  RectanglePtr r,
                           Word cornerDiam)
```

Parameters

r	Pointer to the rectangle to invert.
cornerDiam	Diameter of rounded corners; zero for square corners.

Result
Returns nothing.

See Also
`WinInvertLine`, `WinInvertRectangleFrame`, `WinDrawRectangle`

WinInvertRectangleFrame

Purpose
Invert a rectangular frame in the draw window.

Prototype
```
void WinInvertRectangleFrame ( FrameType frame,
                               RectanglePtr r)
```

Parameters

frame	Type of frame to invert.
r	Pointer to the rectangular frame to invert.

Result
Returns nothing.

See Also
`WinInvertRectangle`, `WinInvertLine`, `WinDrawRectangleFrame`, `WinEraseRectangleFrame`

WinModal

Purpose	Return TRUE if the specified window is modal.
Prototype	`Boolean WinModal (WinHandle winHandle)`
Parameters	winHandle Handle of a window.
Result	Returns TRUE if modal, otherwise FALSE.
Comments	A window is modal if it cannot lose the focus.

WinRemoveWindow

Purpose	Remove the specified window from the window list.
Prototype	`void WinRemoveWindow (WinHandle winHandle)`
Parameters	winHandle Handle of a window.
Result	Returns nothing.
Comments	Does not free the memory used by the window.
See Also	`WinAddWindow`, `WinDeleteWindow`, `WinGetFirstWindow`

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.
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.

Prototype

```
void WinRestoreBits (  WinHandle winHandle,
                       SWord destX, SWord 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`

WinSaveBits

Purpose: Create an offscreen window and copy the specified region from the draw window to the offscreen window.

Prototype

```
WinHandle WinSaveBits (  RectanglePtr sourceP,
                         WordPtr error)
```

Parameters

sourceP	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.

See Also: `WinRestoreBits`

WinScrollRectangle

Purpose Scroll a rectangle in the draw window.

Prototype
```
void WinScrollRectangle (RectanglePtr r,
                         DirectionType direction,
                         SWord distance,
                         RectanglePtr vacated)
```

Parameters

r	Pointer to the rectangle to scroll.
direction	Direction to scroll (up, down, left, or right).
distance	Distance to scroll in pixels.
vacated	Pointer to the rectangle that needs to be redrawn 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.
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. All user input is directed to the active window.
See Also	`WinAddWindow`, `WinGetActiveWindow`

WinSetClip

Purpose	Set the clipping rectangle of the draw window.
Prototype	`void WinSetClip (RectanglePtr r)`
Parameters	r Pointer to a structure holding the clipping bounds.
Result	Returns nothing.
See Also	`WinClipRectangle`, `WinSetClip`, `WinGetClip`

WinSetDrawWindow

Purpose	Set the draw window. (All drawing operations are relative to the draw window.)
Prototype	`WinHandle WinSetDrawWindow (WinHandle winHandle)`
Parameters	winHandle Handle of a window.
Result	Returns the draw window.
See Also	`WinGetDrawWindow, WinSetActiveWindow`

WinSetPattern

Purpose	Set the current fill pattern.
Prototype	`void WinSetPattern (CustomPatternType pattern)`
Parameters	pattern Pattern to use.
Result	Returns nothing.
Comments	The fill pattern is used by `WinFillLine` and `WinFillRectangle`.
See Also	`WinGetPattern`

WinSetUnderlineMode

Purpose	Set the graphic state to enable or disable the underlining of characters.
Prototype	`UnderlineModeType WinSetUnderlineMode (UnderlineModeType mode)`
Parameters	mode — New underline mode type, one of `noUnderline`, `grayUnderline`, `solidUnderline`.
Result	Returns the previous underline mode type.
See Also	`WinDrawChars`

WinWindowToDisplayPt

Purpose	Convert a window-relative coordinate to a display-relative coordinate.
Prototype	`void WinWindowToDisplayPt ( SWordPtr extentX, SWordPtr 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`

11

Miscellaneous User Interface Functions

Category Functions

CategoryCreateList

Purpose	Read a database’s categories and set categories.

Prototype

```
void CategoryCreateList (  DmOpenRef db,
                           ListPtr lst,
                           Word currentCategory,
                           Boolean showAll)
```

Parameters	db	Database containing categories to extract.
	lst	List object to load categories into.
	currentCategory	Will be set as the current selection in the resulting list.
	showAll	TRUE if an “All” category should be included in the list.

Result	Returns nothing.

CategoryEdit

Purpose	Event handler for the Edit Categories dialog.

Prototype

```
Boolean CategoryEdit ( DmOpenRef db,
                       WordPtr category)
```

Parameters

db	Database containing the categories to be edited.
category	Current 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

CategoryFind

Purpose: Return the index of the category that matches the name passed.

Prototype

```
Word CategoryFind (DmOpenRef db, CharPtr name)
```

Parameters

db	Database to search for the passed category.
name	Category name.

Result: Returns the category index.

CategoryFreeList

Purpose	Unlock or free memory locked or allocated by `CategoryCreateList` which was attached to the passed List object.
Prototype	`void CategoryFreeList (DmOpenRef db, ListPtr lst)`
Parameters	db — Database containing the categories.
	1st — Pointer to the category list containing the memory to be freed.
Result	Returns nothing.
Comment	Calling this function does not remove the categories from the passed database.

CategoryGetName

Purpose: Return the name of the specified category.

Prototype:

```
void CategoryGetName ( DmOpenRef db,
                       Word index,
                       CharPtr name)
```

Parameters:

db	Database that contains the categories.
index	Category index.
name	Buffer to hold category name. Buffer should be `dmCategoryLength` in size.

Result: Stores the category name in the `name` buffer passed.

CategoryGetNext

Purpose	Given a category index this routine return the index of the next category. Categories are not stored sequentially.	
Prototype	`Word CategoryGetNext (DmOpenRef db, Word index)`	
Parameters	db	Database that contains the categories.
	index	Category index.
Result	Category index of next category.	

CategoryTruncateName

Purpose Truncate a category name so that it's short enough to display.

Prototype

```
void CategoryTruncateName (  CharPtr name,
                             Word maxWidth)
```

Parameters	name	Category name to truncate.
	maxWidth	Maximum size, in pixels, of truncated category (including ellipsis).

Result Returns nothing

CategorySetTriggerLabel

Purpose Set the label displayed by the category trigger. The category name is truncated if it's to long.

Prototype

```
void CategorySetTriggerLabel ( ControlPtr ctl,
                               CharPtr name)
```

Parameters	ctl	Pointer to control object to relabel.
	label	Pointer to the name of the new category.

Result Returns nothing.

CategorySelect

Purpose — Process the selection and editing of categories.

Prototype

```
Boolean CategorySelect ( DmOpenRef db,
                         FormPtr frm,
                         Word ctlID,
                         Word lstID,
                         Boolean title,
                         WordPtr categoryP,
                         CharPtr categoryName)
```

Parameters

db	Database that contains the categories.
frm	Form that contains the category popup list.
ctlID	ID of the popup trigger.
lstID	ID of the popup list.
title	True if the popup trigger is on the title line.
categoryP	Current category (index into db structure).
categoryName	Name of the current 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

Character Attribute Functions

GetCharAttr

Purpose: Return a pointer to the characters attributes array which is used by the character classification and character conversion macros (such as isalpha and toascii).

Prototype: `WordPtr GetCharAttr (void)`

Parameters: None

Result: A pointer to the attributes array. See CharAttr.h for an explanation of the attributes.

GetCharCaselessValue

Purpose: Return a pointer to an array that maps all characters to an assigned caseless and accentless value. This should be used for finding text.

Prototype: `BytePtr GetCharCaselessValue (void)`

Parameters: None.

Result: A pointer to the sort array.

The compiler pads each byte out to a word so each index position contains two characters.

Note: array[x].high = sort value for character 2x+1.

GetCharSortValue

Purpose	Return a pointer to an array that maps all characters to an assigned sorting value. This should be used for ordering (sorting) text.
Prototype	`BytePtr GetCharSortValue (void)`
Parameters	None.
Result	Returns a pointer to the attributes array. The compiler pads each byte out to a word so each index position contains two characters. Note: array[x].low = sort value for character 2x.

ClipBoard Functions

ClipboardAddItem

Purpose

Add the item passed to the specified clipboard. The `format` parameter determines which clipboard (text, ink, etc.) the item is added to.

Prototype

```
void ClipboardAddItem (ClipboardFormatType format,
                       VoidPtr ptr,
                       Word length)
```

Parameters

format	Text, ink, bitmap, etc.
ptr	Pointer to the item to place on the clipboard.
length	Size of the item to place on the clipboard.

Result

Returns nothing.

See Also

`FldCut`, `FldCopy`

ClipboardGetItem

Purpose Return the handle of the contents of the clipboard of a specified type and the length of a clipboard item.

Prototype
```
VoidHand ClipboardGetItem
      (ClipboardFormatType format, WordPtr length)
```

Parameters

format	Text, ink, bitmap, etc.
length	Pointer to the length of the clipboard item.

Result Handle of the clipboard item.

Font Functions

FntAccentHeight

Purpose Return the height of an accent of the characters in the current font. The height of an accent is the distance between the top of the character cell and the top a non-accent capital letter.

Prototype
```
short FntAccentHeight (void)
```

Parameters None.

Result Height of an accent (in pixels).

FntAscent

Purpose Return the ascent of the characters in the current font. The ascent of a character is the distance from the top of a non-accent capital letter to the base line.

Prototype
```
short FntAscent (void)
```

Parameters None.

Result Returns character ascent (in pixels).

FntAverageCharWidth

Purpose	Return the average character width in the current font.
Prototype	`short FntAverageCharWidth (void)`
Parameters	None.
Result	Returns the average character width (in pixels).

FntBaseLine

Purpose	Return the distance from the top of character cell to the baseline for the current font.
Prototype	`short FntBaseLine (void)`
Parameters	None.
Result	Returns the baseline of the font (in pixels).

FntCharHeight

Purpose	Return the character height, in the current font including accents and descenders.
Prototype	`short FntCharHeight (void)`
Parameters	None
Result	Height of the characters in the current font, expressed in pixels.

FntCharsInWidth

Purpose

Find the number of characters in a string that fit within a passed width. Spaces at the end of a string are ignored and removed. Any characters after a carriage return are ignored and the string is considered truncated.

Prototype

```
void FntCharsInWidth ( CharPtr string,
                       Int *stringWidthP,
                       Int *stringLengthP,
                       Boolean *fitWithinWidth)
```

Parameters

string	Pointer to the char string.
stringWidthP	Maximum width to allow.
stringLengthP	Maximum characters to allow (assumes current Font).
fitWithinWidth	Set to TRUE if the string is considered truncated.

Result

When the call is completed, the information is updated as follows:

stringWidthP	Set to the width of the chars allowed.
stringLengthP	Set to the number of chars within the width.
fitWithinWidth	TRUE if the string is considered truncated, FALSE if it isn’t.

FntCharsWidth

Purpose	Return the width of the specified character string. The Missing Character Symbol is substituted for any character which does not exist in the current font.	
Prototype	`short FntCharsWidth (CharPtr pChars, Word length)`	
Parameters	pChars	Pointer to a string of characters.
	length	Number of character in the string.
Result	Returns the width of the string, in pixels.	

FntCharWidth

Purpose	Return the width of the specified character. If the specified character does not exist within the current font, the Missing Character Symbol is substituted.	
Prototype	`short FntCharWidth (char ch)`	
Parameters	ch	Character whose width is needed.
Result	Returns the width of the specified character (in pixels).	

FntDescenderHeight

Purpose	Return the height of a character's descender in the current font. The height of a descender is the distance between the base line an the bottom of the character cell.
Prototype	`short FntDescenderHeight (void)`
Parameters	None.
Result	Returns the height of a descender, expressed in pixels.

FntGetFont

Purpose	Return the Font ID of the current font.
Prototype	`FontID FntGetFont (void)`
Parameters	None.
Result	Returns FontID of the current font.

FntGetFontPtr

Purpose	Return a pointer to the current font.
Prototype	`FontPtr FntGetFontPtr (void)`
Parameters	None.
Result	Returns the FontPtr of the current font.

FntLineHeight

Purpose	Return 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).
Prototype	`short FntLineHeight (void)`
Parameters	None.
Result	Returns the height of a line in the current font.

FntLineWidth

Purpose	Return the width of the specified line of text, taking tab characters in to 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.
Prototype	`short FntLineWidth (CharPtr pChars, Word length)`
Parameters	pChars — Pointer to a string of characters. length — Number of character in the string.
Result	Returns the line width (in pixels).

FntProportionalFont

Purpose	Indicate whether the current font is proportionally spaced or fixed width.
Prototype	`Boolean FntProportionalFont (void)`
Parameters	None.
Result	Returns TRUE if the current font is proportionally spaced, FALSE if it's fixed width.

FntSetFont

Purpose	Set the current font.
Prototype	`FontID FntSetFont (FontID fontID)`
Parameters	fontID — ID of the font to make the active font.
Result	Returns ID of the current font before the change.

Other User Interface Functions

AbtShowAbout

Purpose	Displays the info dialog box. The application name is picked up from either the tAIN resource of the application, or the name of the application database (which is assigned in the makefile).
Prototype	`void AbtShowAbout (ULong creator)`
Parameters	creator — Creator ID of this application.
Result	Returns nothing.

DayHandleEvent

Purpose: Handle event in the specified control. This routine handles two type of events, penDownEvent and ctlEnterEvent.

Prototype:

```
Boolean DayHandleEvent ( DaySelectorPtr pSelector,
                         EventPtr pEvent)
```

Parameters:

pSelector	Pointer to control object (ControlType)
pEvent	Pointer to an EventType structure.
pError	Pointer to returned error code

Result: True if the event was handle or false if it was not.

Posts a daySelectEvent with info on whether to use the date.

A date is used if the user selects a day in the visible month.

Functions for System Use Only

Find

Prototype:

```
void Find (GoToParamsPtr goToP)
```

WARNING: System Use Only!

FindDrawHeader

Prototype
```
Boolean FindDrawHeader ( FindParamsPtr params,
                         CharPtr title)
```

WARNING: System Use Only!

FindGetLineBounds

Prototype
```
void FindGetLineBounds ( FindParamsPtr params,
                         RectanglePtr r)
```

WARNING: System Use Only!

FindSaveMatch

Prototype
```
Boolean FindSaveMatch (  FindParamsPtr params,
                         UInt recordNum,
                         Word pos,
                         UInt fieldNum,
                         DWord appCustom,
                         UInt dbCardNo,
                         LocalID dbID)
```

WARNING: System Use Only!

FindStrInStr

Prototype
```
Boolean FindStrInStr ( CharPtr strToSearch,
                       CharPtr strToFind,
                      WordPtr posP)
```

WARNING: System Use Only!

UIInitialize

Prototype `void UIInitialize (void)`

WARNING: System Use Only!

UIReset

Prototype `void UIReset (void)`

WARNING: System Use Only!

12

System, Error, Preferences, and Find Functions

System Functions

SysAppLaunch

Purpose Launch the specified application with the given command line arguments, given a card number and database ID of an application resource database.

Prototype

```
Err SysAppLaunch( UInt cardNo, LocalID dbID,
                  UInt launchFlags, Word cmd,
                  Ptr cmdPBP, DWord* resultP)
```

Parameters

cardNo, dbID	`cardNo` and `dbID` identify the application.
launchFlags	Set to 0.
cmd	Action code.
cmdPBP	Action code parameter block.
resultP	Pointer to what's returned by the application's `PilotMain` routine.

Result Returns 0 if no error, or one of `sysErrParamErr`, `memErrNotEnoughSpace`, `sysErrOutOfOwnerIDs`.

Comments Launching an application with all launch bits cleared makes the application a subroutine call from the point of view of the caller.

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!

Prototype

```
UInt SysBatteryInfo(   Boolean set,
                       UIntPtr warnThresholdP,
                       UIntPtr criticalThresholdP,
                       UIntPtr maxTicksP,
                       SysBatteryKind* kindP,
                       Boolean* pluggedIn)
```

Parameters

set	If false, parameters with non-nil pointers are retrieved. Never set this parameter to TRUE.
warnThresholdP	Pointer to battery voltage warning threshold in volts*100, or nil.
criticalThresholdP	Pointer to the battery voltage critical threshold in volts*100, or nil.
maxTicksP	Pointer to the battery timeout, or nil.
kindP	Pointer to the battery kind, or nil.
pluggedInP	Pointer to `pluggedIn` return value, or nil.

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 `SysBatteryWarningDialog` 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.

SysBroadcastActionCode

Purpose	Send the specified action code and parameter block to the latest version of every UI application.
Prototype	`Err SysBroadcastActionCode (Word cmd, Ptr 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, sysErrOutOfOwnerIDs.`
See Also	`SysAppLaunch`

SysCopyStringResource

Purpose
: Copy a resource string to a passed string.

Prototype

```
void SysCopyStringResource ( CharPtr string,
                             UInt theID)
```

Parameters

string	String to copy the resource string to
theID	Resource string ID

Result
: Stores a copy of the resource string in `string`.

SysCurAppDatabase

Purpose	Return the card number and database ID of the current application's resource database.

Prototype

```
Err SysCurAppDatabase (  UIntPtr cardNoP,
                         LocalID* dbIDP)
```

Parameters

cardNoP	Pointer to the card number; 0 or 1.
dbIDB	Pointer to the database ID.

Result: Returns 0 if no error, or `SysErrParamErr` if an error occurs.

See Also: `SysAppLaunch`, `SysUIAppSwitch`

SysFormPointerArrayToStrings

Purpose: Form an array of pointers to strings in a block. Useful for setting the items of a list.

Prototype

```
VoidHand SysFormPointerArrayToStrings
                       (CharPtr c,
                       Int stringCount)
```

Parameters

c	Pointer to packed block of strings, each terminated by NULL.
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.

SysHandleEvent

Purpose	Handle defaults for system events such as hard and soft key presses.
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. 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 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.

Prototype

```
void SysInsertionSort (Byte baseP,
                       Int numOfElements,
                       Int width,
                       CmpFuncPtr comparF,
                       Long 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).
other	Other data passed to the comparison function.

Result

Returns nothing.

Comments

The comparison function (`comparF`) has this prototype:

```
  int comparF (BytePtr A, BytePtr B, Long other);
```

The function returns:

- > 0 if A > B
- < 0 if A< B
- 0 if A = B

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.

Prototype `void SysKeyboardDialog ()`

Parameters None.

Result Returns nothing. The field's text chunk is changed.

See Also `FrmSetFocus`

SysQSort

Purpose Sort elements in an array according to the passed comparison function. 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 needing moving.

Prototype

```
void SysQSort ( Byte baseP,
                Int numOfElements,
                Int width,
                CmpFuncPtr comparF,
                Long 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 for `SysInsertionSort`.
other	Other data passed to the comparison function.

Result Returns nothing.

See Also `SysInsertionSort`

SysRandom

Purpose Return a random number anywhere from 0 to `sysRandomMax`.

Prototype `Int SysRandom (ULong 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.

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 button 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.

Prototype
`UInt   SysSetAutoOffTime (UInt seconds)`

Parameters

seconds	Time out in seconds, or 0 for no time out

Result
Returns previous value of time out in seconds.

SysTaskDelay

Purpose	Put the processor into doze mode for the specified number of ticks.
Prototype	`Err SysTaskDelay (Long delay)`
Parameters	delay — Number of ticks to wait (see sysTicksPerSecond)
Result	Returns 0 if no error.
See Also	`EvtGetEvent`

SysUIAppSwitch

Purpose: Try to make the current UI application quit and then launch the UI application specified by card number and database ID.

Prototype:

```
Err SysUIAppSwitch(  UInt cardNo,
                     LocalID dbID,
                     Word cmd,
                     Ptr cmdPBP)
```

Parameters:

cardNo	Card number for the new application; currently only card 0 is valid.
dbID	ID of the new application.
cmd	Action code.
cmdPBP	Action code parameter block.

Result: Returns 0 if no error.

See Also: `SysAppLaunch`

Functions for System Use Only

SysAppExit

Prototype
```
Err SysAppExit (SysAppInfoPtr appInfoP,
      Ptr prevGlobalsP, Ptr globalsP)
```

WARNING: System Use Only!

SysAppInfoPtr

Prototype
```
SysAppInfoPtr SysCurAppInfoP (void)
```

WARNING: System Use Only!

SysAppStartup

Prototype
```
Err   SysAppStartup (  SysAppInfoPtr appInfoPP,
      Ptr prevGlobalsP, Ptr globalsP)
```

WARNING: System Use Only!

SysBatteryDialog

Prototype
```
void SysBatteryDialog (void)
```

WARNING: System Use Only!

SysCardImageDeleted

Prototype
```
void SysCardImageDeleted (UInt cardNo)
```

WARNING: System Use Only!

SysCardImageInfo

Prototype `Ptr SysCardImageInfo (UInt cardNo, ULongPtr sizeP)`

WARNING: System Use Only!

SysColdBoot

Purpose Perform a cold boot and reformat all RAM areas of both memory cards.

WARNING: System Use Only!

SysCurAppInfoP

Prototype `SysCurAppInfoPtr SysCurrAppInfoP (void)`

WARNING: System Use Only!

SysDisableInts

Prototype `Word SysDisableInts (void)`

WARNING: System Use Only!

SysDoze

Prototype `void SysDoze (Boolean onlyNMI)`

WARNING: System Use Only!

SysGetTrapAddress

Prototype `VoidPtr SysGetTrapAddress (UInt trapNum)`

WARNING: System Use Only!

SysInit

Prototype `void SysInit (void)`

WARNING: System Use Only!

SysKernelInfo

Prototype `Err SysKernelInfo (VoidPtr paramP)`

WARNING: System Use Only!

SysLaunchConsole

Prototype `Err SysLaunchConsole (void)`

WARNING: System Use Only!

SysLibFind

Prototype `Err SysLibFind (CharPtr nameP, UIntPtr refNumP)`

WARNING: System Use Only!

SysLibInstall

Prototype
```
Err   SysLibInstall ( SysLibEntryProcPtr libraryP,
                      UIntPtr refNumP)
```

WARNING: System Use Only!

SysLibRemove

Prototype
```
Err SysLibRemove (UInt refNum)
```

WARNING: System Use Only!

SysLibTblEntry

Prototype
```
SysLibTblEntryPtr SysLibTblEntry (UInt refNum)
```

WARNING: System Use Only!

SysNewOwnerID

Prototype
```
UInt SysNewOwnerID (void)
```

WARNING: System Use Only!

SysPowerOn

Prototype
```
void SysPowerOn (  Ptr card0P, ULong card0Size,
                   Ptr card1P, ULong card1Size,
                   DWord sysCardHeaderOffset,
                   Boolean reFormat)
```

WARNING: System Use Only!

SysRestoreStatus

Prototype
```
void SysRestoreStatus (Word status)
```

WARNING: System Use Only!

SysSetA5

Prototype
```
DWord SysSetA5 (DWord newValue)
```

WARNING: System Use Only!

SysSetTrapAddress

Prototype
```
Err SysSetTrapAddress (  UInt trapNum,
                         VoidPtr procP)
```

WARNING: System Use Only!

SysSleep

Prototype
```
void SysSleep (  Boolean untilReset,
                 Boolean emergency)
```

WARNING: System Use Only!

SysUILaunch

Prototype
```
void SysUILaunch (void)
```

WARNING: System Use Only!

Error Manager Functions

ErrDisplay

Purpose	Display an error alert if error checking is set to partial or full.	
Prototype	`void ErrDisplay (char* message)`	
Parameters	-> message	Error message text.
Result	No return value.	
Comments	Call this routine to display an error message, source code filename, and line number. This routine is actually a macro that 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`, `“Using the Error Manager”`	

ErrDisplayFileLineMsg

Purpose: Display a nonexitable dialog with an error message. Do not allow the user to continue.

Prototype:

```
void ErrDisplayFileLineMsg(  CharPtr filename,
                             UInt lineno,
                             CharPtr 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.

See Also — `ErrFatalDisplayIf`, `ErrNonFatalDisplayIf`, `ErrDisplay`

ErrFatalDisplayIf

Purpose — Display an error alert dialog if `condition` is TRUE and error checking is set to partial or full.

Prototype

```
void ErrFatalDisplayIf ( Boolean condition,
                         char* message)
```

Parameters

-> condition	If TRUE, display the error.
-> message	Error message text.

Result — No return value.

Comments — Call this routine 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 routine is actually a macro that 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`, `"Using the Error Manager"`

ErrNonFatalDisplayIf

Purpose
: Display an error alert dialog if `condition` is TRUE and error checking is set to full.

Prototype

```
void ErrNonFatalDisplayIf (  Boolean condition,
                             char* message)
```

Parameters

-> condition	If TRUE, display the error.
-> message	Error message text.

Result
: No return value.

Comments
: Call this routine 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 routine is actually a macro that is compiled into the code only if the compiler define ERROR_CHECK_LEVEL is set to 2 (ERROR_CHECK_FULL).

See Also
: `ErrFatalDisplayIf`, `ErrDisplay`, `"Using the Error Manager"`

ErrThrow

Purpose
: Cause a jump to the nearest Catch block.

Prototype

```
void ErrThrow (Long 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`, `"Using the Error Manager"`

System Preferences Functions

PrefGetAppPreferences

Purpose	Return a copy of an application's preferences.

Prototype

```
Boolean PrefGetAppPreferences (ULong type,
                               Int version,
                               VoidPtr prefs,
                               Word prefsSize)
```

Parameters

type	Application creator type.
version	Version number of the application.
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.

See Also: `PrefSetPreferences`

PrefGetPreferences

Purpose	Return a copy of the system preferences.
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.
See Also	`PrefSetPreferences`

PrefOpenPreferenceDB

Purpose	Return a handle to the system preference database.
Prototype	`DmOpenRef PrefOpenPreferenceDB (void)`
Parameters	Nothing.
Result	Returns the handle, or 0 if an error results.
See Also	`PrefGetPreferences`, `PrefSetPreferences`

PrefSetAppPreferences

Purpose	Save an application's preferences in the preferences database.

Prototype
```
void PrefSetAppPreferences ( ULong type,
                             Int version,
                             VoidPtr prefs,
                             Word prefsSize)
```

Parameters

type	Application creator type.
version	Version number of the application.
prefs	Pointer to a buffer holding preferences.
prefsSize	Size of the buffer passed.

Result: Nothing.

Comments: The content and format of an application preference is application-dependent.

See Also: `PrefGetPreferences`

PrefSetPreferences

Purpose: Set the system preferences.

Prototype
```
void PrefSetPreferences (SystemPreferencesPtr p)
```

Parameters

p	Pointer to system preferences.

Result: Returns nothing.

See Also: `PrefGetPreferences`

Find Functions

FindDrawHeader

Purpose	Draw the header line that separates, by database, the list of found items.

Prototype

```
Boolean FindDrawHeader ( FindParamsPtr params,
                         CharPtr title)
```

Parameters

params	Handle of `FindParamsPtr`.
title	Description of the database (for example Memos)

Result — Returns TRUE if Find screen is filled up. Applications should exit from the search if this occurs.

FindGetLineBounds

Purpose — Returns the bounds of the next available line for displaying a match in the Find Results dialog.

Prototype

```
void FindGetLineBounds ( FindParamsPtr params,
                         RectanglePtr r)
```

Parameters

params	Handle of `FindParamsPtr`.
r	Pointer to a structure to hold the bounds of the next results line.

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.

Prototype
```
void FindSaveMatch    ( FindParamsPtr params,
                        UInt recordNum,
                        Word pos,
                        UInt fieldNum,
                        DWord appCustom,
                        UInt dbCardNo,
                        LocalID rdbID)
```

Parameters

params	Handle of `FindParamsPtr`.
recordNum	Record index.
pos	Offset of the match string from start of record.
appCustom	Extra data the application can save with a match.
dbCardNo	Car number of the database that contains the match.
dbID	Local ID of the database that contains the match.

Result Returns TRUE if the maximum number of displayable items has been exceeded

Comments Called by application code when it gets a match.

FindStrInStr

Purpose

Perform a case-blind partial word search for a string in another string. This function assumes that the string to find is in lower-case characters.

Prototype

```
void FindStrInStr      ( CharPtr strToSearch,
                         CharPtr strToFind,
                         WordPtr posP)
```

Parameters

strToSearch	String to search.
strToFind	String to find.
posP	Pointer to the offset in the search string of the match.

Result

Returns TRUE if the string was found.

13

System Event Manager Functions

EvtAddEventToQueue

Purpose Add an event to the event queue.

Prototype `void EvtAddEventToQueue (EventPtr event)`

Parameters

event	Pointer to the structure that contains the event.
error	Pointer to any error encountered by this function.

Result Returns nothing.

EvtCopyEvent

Purpose Copy an event.

Prototype `void EvtCopyEvent (EventPtr source, EventPtr dest)`

Parameters

source	Pointer to the structure containing the event to copy.
dest	Pointer to the structure to copy the event to.

Result Returns nothing.

EvtDequeuePenPoint

Purpose Get the next pen point out of the pen queue (called by the recognizers).

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`.

See Also `EvtDequeuePenStrokeInfo`

EvtDequeuePenStrokeInfo

Purpose Initiate the extraction of a stroke from the pen queue.

Prototype

```
Err EvtDequeuePenStrokeInfo( PointType* startPtP,
                             PointType* endPtP)
```

Parameters

startPtP	Start point returned here.
startPtP	End point returned here.

Result Always returns 0.

Comments Called by the system function `EvtGetSysEvent`. 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.

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.

Prototype

```
Err EvtEnqueueKey (  UInt ascii,
                     UInt keycode,
                     UInt modifiers)
```

Parameters

ascii	ascii code of key.
keycode	Virtual key code of key.
modifiers	Modifiers for key event.

Result Returns 0 if successful, or `evtErrParamErr` if an error occurs.

Comments Called by the keyboard interrupt routine and the Graffiti and Soft-Keys recognizers. Note that because both interrupt- and noninterrupt-level code can post keys into the queue, this routine disables interrupts while the queue header is being modified.

Most keys in the queue take only 1 byte if they have no modifiers and no virtual key code, and are 8-bit ASCII. If a key event in the queue has modifiers or is a non-standard ascii code, it takes up to 7 bytes of storage and has the following format:

evtKeyStringEscape	1 byte
ASCII code	2 bytes
virtual key code	2 bytes
modifiers	2 bytes

EvtFlushKeyQueue

Purpose Flush all keys out of the key queue.

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.

Prototype `Err EvtFlushNextPenStroke (void)`

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 de-queueing. Otherwise, this routine flushes the next stroke in the queue.

See Also `EvtDequeuePenPoint`

EvtFlushPenQueue

Purpose	Flush all points out of the pen queue.
Prototype	`Err EvtFlushPenQueue (void)`
Parameters	None
Result	Always returns 0.
Comment	Called by the system function `EvtSetKeyQueuePtr`.
See Also	`EvtPenQueueSize`

EvtGetEvent

Purpose: Return the next available event.

Prototype:

```
void EvtGetEvent (EventPtr event, Long timeout)
```

Parameters:

event	Pointer to the structure to hold the event returned.
timeout	Max amount of ticks to wait before an event is returned (-1 means wait indefinitely).

Comments: Pass `timeout`= -1 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 `timeout` >= 0.

Result: Returns nothing.

EvtGetPen

Purpose	Return the current status of the pen.	
Prototype	`void EvtGetPen( Sword *pScreenX, Sword *pScreenY, Boolean *pPenDown)`	
Parameters	pScreenX	x location relative to display.
	pScreenY	y location relative to display.
	pPenDown	TRUE or FALSE.
Result	Returns nothing.	
Comments	Called by various UI routines.	
See Also	`KeyCurrentState`	

EvtGetPenBtnList

Purpose	Return a pointer to the silk-screen button array.	
Prototype	`PenBtnInfoPtr asm EvtGetPenBtnList( UIntPtr numButtons)`	
Parameters	numButtons	Pointer to the variable to contain the number of buttons in the array.
Result	Returns a pointer to the array.	
Comments	The array returned contains the bounds of each silk-screened button and the ASCII code and modifiers byte to generate for each button.	
See Also	`EvtProcessSoftKeyStroke`	

EvtKeyQueueEmpty

Purpose	Return TRUE if the key queue is currently empty.
Prototype	`Boolean EvtKeyQueueEmpty (void)`
Parameters	None.
Result	Returns TRUE if the key queue is currently empty, otherwise returns FALSE.
Comments	Called by key manager to determine if it should enqueue auto-repeat keys.

EvtKeyQueueSize

Purpose	Return the size of the current key queue in bytes.
Prototype	`ULong 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.
Prototype	`ULong 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 in to the key queue.

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 to assure that the device doesn't automatically power off during a long operation without user input (for example, serial port activity).

Prototype

```
Err EvtResetAutoOffTimer (void)
```

Parameters None.

Result Always returns 0.

Comments Called by `SerialLinkMgr`, Can be called periodically by other managers.

See Also `SysSetAutoOffTime`

EvtWakeup

Purpose Force the event manager to wake up and send a `nilEvent` to the current application.

Prototype
```
Err EvtWakeup (void)
```

Parameters None.

Result Always returns 0.

Comments Called by interrupt routines, like the sound manager and alarm manager.

Functions for System Use Only

EvtDequeueKeyEvent

Prototype
```
Err EvtDequeueKeyEvent (EventPtr eventP)
```

WARNING: System Use Only!

EvtEnqueuePenPoint

Prototype
```
Err EvtEnqueuePenPoint (PointType* ptP)
```

WARNING: System Use Only!

EvtGetSysEvent

Prototype
```
void EvtGetSysEvent (  EventPtr eventP,
                       Long timeout)
```

WARNING: System Use Only!

EvtInitialize

Prototype `void EvtInitialize (void)`

WARNING: System Use Only!

EvtSetKeyQueuePtr

Prototype `Err EvtSetKeyQueuePtr (Ptr keyQueueP, ULong size)`

WARNING: System Use Only!

EvtSetPenQueuePtr

Prototype `Err EvtSetPenQueuePtr (Ptr penQueueP, ULong size)`

WARNING: System Use Only!

EvtSysInit

Prototype `Err EvtSysInit (void)`

WARNING: System Use Only!

14

Feature, Time, Float, and String Functions

FtrGet

Purpose	Get a feature.	
Prototype	`Err FtrGet ( DWord creator, UInt featureNum, DWordPtr valueP)`	
Parameters	creator	Creator type, should be same as 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 `ftrErrNoSuchFtr` or `ftrErrInternalError` if an error occurs.	
Comments	The value of the feature is application-dependent.	
See Also	`FtrSet`	

FtrGetByIndex

Purpose

Get a feature by index.

Until the caller gets back `ftrErrNoSuchFeature`, it should pass indices for each table (ROM, RAM) starting at 0 and incrementing .

Prototype

```
Err FtrGetByIndex (  UInt index,
                     Boolean romTable,
                     DWordPtr creatorP,
                     UIntPtr numP,
                     DWordPtr 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 `ftrErrInternalError` or `ftrErrNoSuchFeature` if an error occurs.

Comments

This routine is normally only used by shell commands. Most applications do not need it.

FtrSet

Purpose	Set a feature.

Prototype

```
Err FtrSet ( DWord creator,
             UInt featureNum,
             DWord newValue)
```

Parameters

creator	Creator type, should be same as the application that owns this feature.
featureNum	Feature number of the feature.
newValue	New value.

Result: Returns 0 if no error, or `ftrErrNoSuchFeature`, `memErrChunkLocked`, `memErrInvalidParam`, or `memErrNotEnoughSpace` if an error occurs.

Comments: The value of the feature is application-dependent.

See Also: `FtrGet`

FtrUnregister

Purpose: Unregister a feature.

Prototype

```
Err FtrUnregister (DWord creator,
                   UInt featureNum)
```

Parameters

creator	Creator type, should be same as the application that owns the creator.
featureNum	Feature number of the feature.

Result: Returns 0 if no error, or `ftrInternalError`, `ftrErrNoSuchFeature`, `memErrChunkLocked`, `memErrInvalidParam`, or `memErrNotEnoughSpace` if an error occurs.

For System Use Only

FtrInit

Prototype `Err FtrInit (void)`

WARNING: This function for System use only

String Manager Functions

StrAToI

Purpose Converts a string to an integer.

Prototype `Int StrAToI (CharPtr str)`

Parameters str String to convert.

Result Returns the integer.

Comments Use this function instead of the standard `atoi` routine.

StrCat

Purpose Concatenate one string to another.

Prototype `CharPtr StrCat (CharPtr dst, CharPtr src)`

Parameters Two string pointers.

Result Returns a pointer to the destination string.

Comments Use this function instead of the standard `strcat` routine.

StrCaselessCompare

Purpose	Compare two strings with case and accent insensitivity.
Prototype	`Int StrCaselessCompare (CharPtr s1, CharPtr s2)`
Parameters	Two string pointers.
Result	Returns 0 if the two strings match, or non-zero if they don't.
Comments	Use this function instead of the standard `stricmp` routine. Use it to find strings but not sort them because it ignores case and accents.
See Also	`StrCompare`

StrChr

Purpose	Look for a character within a string.	
Prototype	`CharPtr StrChr (CharPtr str, Int chr)`	
Parameters	str	String to search.
	chr	Character to search for.
Result	Returns a pointer to the first occurrence of character in `str`, or NULL if not found.	
Comments	Use this function instead of the standard `strchr` routine. This routine does not correctly find a '\0' character.	
See Also	`StrStr`	

StrCompare

Purpose	Compare two strings.
Prototype	`Int StrCompare (CharPtr s1, CharPtr s2)`
Parameters	Two string pointers.
Result	Returns 0 if the strings match. Returns a positive number if s1 > s2. Returns a negative number if s1 < s2.
Comments	This function is case sensitive. Use it to sort strings but not to find them. Use this function instead of the standard `strcmp` routine.
See Also	`StrCaselessCompare`

StrCopy

Purpose	Copy one string to another.
Prototype	`CharPtr StrCopy (CharPtr dst, CharPtr src)`
Parameters	Two string pointers.
Result	Returns a pointer to the destination string.
Comments	Use this function instead of the standard `strcpy` routine. This function does not return overlapping strings.

StrIToA

Purpose	Convert an integer to ASCII.
Prototype	`CharPtr StrIToA (CharPtr s, Long i)`
Parameters	s String pointer to store 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.
Prototype	`CharPtr StrIToH (CharPtr s, ULong i)`
Parameters	s String pointer to store results.
	i Integer to convert.
Result	Returns the string pointer `s`.
See Also	`StrIToA`

StrLen

Purpose	Compute the length of a string.
Prototype	`UInt StrLen (CharPtr src)`
Parameters	src String pointer
Result	Returns the length of the string.
Comments	Use this function instead of the standard `strlen` routine.

StrStr

Purpose	Look for a substring within a string.
Prototype	`CharPtr StrStr (CharPtr str, CharPtr token)`
Parameters	str — String to search. token — 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.
See Also	`StrChr`

StrToLower

Purpose	Convert all the characters in a string to lowercase.
Prototype	`CharPtr StrToLower (CharPtr dst, CharPtr src)`
Parameters	Two string pointers.
Result	Returns a pointer to the destination string.
Comments	This function **doesn't** convert accented characters.

Time Manager Functions

DateAdjust

Purpose	Return a new date +/- the days adjustment.	
Prototype	`void DateAdjust (DatePtr dateP, Long adjustment)`	
Parameters	dateP	A `DateType` structure with the date to be adjusted (see DateTime.h).
	adjustment	The adjustment in seconds.
Result	Changes `dateP` to contain the new date.	
Comments	This function is useful for advancing a day or week and not worrying about month and year wrapping. If the time is advanced out of bounds, it is cut at the bounds surpassed.	

DateDaysToDate

Purpose	Return the date, given days.	
Prototype	`void DateDaysToDate (ULong days, DatePtr dateP)`	
Parameters	days	Days since 1/1/1904.
	dateP	Pointer to `DateType` structure (returned).
Result	Returns nothing, stores the date in `dateP.`	
See Also	`TimAdjust`, `DateToDays`	

DateSecondsToDate

Purpose	Return the date given seconds.

Prototype

```
void DateSecondsToDate ( ULong seconds,
                         DatePtr dateP)
```

Parameters

seconds	Seconds since 1/1/1904.
dateP	Pointer to DateType structure (returned).

Result: Returns nothing; stores the date in `dateP`.

DateToAscii

Purpose: Convert the time passed to an ASCII string in the passed `DateFormatType`.

Prototype

```
void DateToAscii(  Byte months,
                   Byte days,
                   Word years,
                   DateFormatType dateFormat,
                   CharPtr pString)
```

NOTE: Handles the long **and** short forms of the date formats.

Parameters

months	Months (1-12).
days	Days (1-31).
years	Years (for example 1995).
dateFormat	Long or short `DateFormatType`.
pString	Pointer to string which gets the result. Must be of length `dateStringLength` for standard formats or `longDateStrLength` for long date formats.

Result: Returns nothing; stores the result in `pString`.

See Also: `TimeToAscii`, `DateToDOWDMFormat`

DateToDays

Purpose	Return the date in days since 1/1/1904.
Prototype	`ULong DateToDays (DateType date)`
Parameters	date `DateType` structure.
Result	Returns the days since 1/1/1904.
See Also	`TimAdjust`, `DateDaysToDate`

DateToDOWDMFormat

Purpose Convert the date passed to an ASCII string.

Prototype

```
void DateToDOWDMFormat( Byte months,
                        Byte days,
                        Word years,
                        DateFormatType dateFormat,
                        CharPtr pString)
```

Parameters

months	Month (1-12).
days	Day (1-31).
years	Years (for example 1995).
dateFormat	False to use AM and PM.
pString	Pointer to string which gets the result. The string must be of length `timeStringLength`.

Result Returns nothing; stores ASCII string in `pString`.

See Also `DateToAscii`

DayOfMonth

Purpose	Return the day of a month on which the specified date occurs (for example, `dom2ndTue`).
Prototype	`UInt DayOfMonth (UInt month, UInt day, UInt year)`
Parameters	month — Month (1-12). day — Day (1-31). year — Year (ex: 1995).
Result	Returns the day of the month as a `DayOfWeekType`, see DateTime.h.

DayOfWeek

Purpose	Return the day of the week.
Prototype	`UInt DayOfWeek (UInt month, UInt day, UInt year)`
Parameters	month — Month (1-12). day — Day (1-31). year — Year (ex: 1995).
Result	Returns the day of the week (Sunday = 0, Monday = 1, etc.).

DaysInMonth

Purpose	Return the number of days in the month.
Prototype	`UInt DaysInMonth (UInt month, UInt year)`
Parameters	month — Month (1-12). year — Year (for example, 1995).
Result	Returns the number of days in the month for that year.

SelectDay

Purpose

Display a form showing a date and allow the user to select a different date.

Prototype

```
Boolean SelectDay (int *month,
                   int *day,
                   int *year,
                   CharPtr title)
```

Parameters

month	Month selected.
day	Day selected.
year	Year selected.
title	String title for the dialog.

Result

Returns true if the OK button was pressed. In that case, the parameters passed are changed.

TimAdjust

Purpose

Return a new date, +/- the time adjustment.

Prototype

```
void TimAdjust( DateTimePtr dateTimeP,
                Long adjustment)
```

Parameters

dateTimeP	A `DateType` structure (see DateTime.h).
adjustment	The adjustment in seconds.

Result

Returns nothing. Changes `dateTimeP` to the new date and time.

Comments

This function is useful for advancing a day or week and not worrying about month and year wrapping.

If the time is advanced out of bounds it is cut at the bounds surpassed.

See Also

`DateAdjust`

TimDateTimeToSeconds

Purpose	Return the date and time in seconds since 1/1/1904.
Prototype	`ULong TimDateTimeToSeconds (DateTimePtr dateTimeP)`
Parameters	dateTimeP — A `DateType` structure (see DateTime.h).
Result	The time in seconds since 1/1/1904.
See Also	`TimSecondsToDateTime`

TimGetSeconds

Purpose	Return seconds since 1/1/1904.
Prototype	`ULong TimGetSeconds (void)`
Parameters	None.
Result	Returns the number of seconds.
See Also	`TimSetSeconds`

TimGetTicks

Purpose	Return the tick count since the last reset.
Prototype	`ULong TimGetTicks (void)`
Parameters	None.
Result	Returns the tick count.
Comments	The tick count does not advance while the device is in sleep mode.

TimSecondsToDateTime

Purpose	Return the date and time, given seconds.

Prototype

```
void TimSecondsToDateTime( ULong seconds,
                           DateTimePtr dateTimeP)
```

Parameters		
	seconds	Seconds to advance from 1/1/1904.
	dateTimeP	A `DateTimeType` structure that's filled by the function.

Result	Returns nothing. Stores the date and time given seconds since 1/1/1904 in `dateTimeP.`
See Also	`TimDateTimeToSeconds`

TimSetSeconds

Purpose	Return seconds since 1/1/1904.

Prototype

```
void TimSetSeconds (ULong seconds)
```

Parameters		
	seconds	Place to return the seconds since 1/1/1904.

Result	Returns nothing; modifies `seconds`.
See Also	`TimGetSeconds`

TimeToAscii

Purpose
: Convert the time passed to an ASCII string.

Prototype

```
void TimeToAscii(  Byte hours,
                   Byte minutes,
                   TimeFormatType timeFormat,
                   CharPtr pString)
```

Parameters

hours	Hours (0-23).
minutes	Minutes (0-59).
timeFormat	False to use AM and PM.
pString	Pointer to string which gets the result. Must be of length `timeStringLength`.

Result
: Returns nothing. Stores pointer to the text of the current selection in `pString`.

See Also
: `DateToAscii`

Functions for System Use Only

TimGetAlarm

Prototype

```
ULong TimGetAlarm (void)
```

WARNING: System use only!

TimHandleInterrupt

Prototype

```
void TimHandleInterrupt (Boolean periodicUpdate)
```

Warning: System use only!

TimInit

Prototype `Err TimInit (void)`

Warning: System use only!

TimSetAlarm

Prototype `ULong TimSetAlarm (ULong alarmSeconds)`

Warning: System use only!

Float Manager Functions

FplAdd

Purpose Add two floating-point numbers (returns a + b).

Prototype `FloatType FplAdd (FloatType a, FloatType b)`

Parameters

a, b	The floating-point numbers.

Result Returns the normalized floating-point result of the addition.

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.
See Also	`FplFToA`

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.

FplDiv

Purpose	Divide two floating-point numbers (result = dividend / divisor).

Prototype

```
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.

FplFloatToLong

Purpose Convert a floating-point number to a long integer.

Prototype

```
Long FplFloatToLong (FloatType f)
```

Parameters

f	Floating-point number to be converted.

Result Returns the long integer.

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.

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.
See Also	`FplInit`

FplFToA

Purpose	Convert a floating-point number to a zero-terminated ASCII string in exponential format : [-]x.yyyyyyyye[-]zz
Prototype	`Err FplFToA (FloatType a, char* s)`
Parameters	a — The floating-point number. s — Pointer to buffer to contain the ASCII string.
Result	Returns an error code, or 0 if no error.
See Also	`FplAToF`

FplInit

Purpose	Initialize the floating-point conversion routines. Allocate space in the system heap for `fpl` 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.
See Also	`FplFree`

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.

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.

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.

Alarm Manager API

AlmGetAlarm

Purpose: Return the alarm date/time in seconds since 1/1/1904 and the caller-defined alarm reference value for the given application.

Prototype:

```
ULong AlmGetAlarm (  UInt cardNo,
                     LocalID dbID,
                     DWordPtr refP)
```

Parameters:

-> cardNo	Storage card number of the application.
-> dbID	Local ID of the application.
<-> refP	Pointer to location for the alarm's reference value.

Result: Alarm 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.

AlmSetAlarm

Purpose
Set or cancel an alarm for the given application.

Prototype
```
Err AlmSetAlarm (  UInt cardNo,
                   LocalID dbID,
                   DWord ref,
                   ULong alarmSeconds,
                   Boolean quiet)
```

Parameters

-> cardNo	Storage card number of the application.
-> dbID	Local ID of the application.
-> ref	Caller-defined value to be passed with notifications.
-> alarmSeconds	Alarm date/time in seconds since 1/1/1904, or 0 to cancel the current alarm (if any).
-> quiet	Reserved for future upgrade (set to zero).

Result

0	No error.
almErrMemory	Insufficient memory.
almErrFull	Alarm table is full.

Comments
If an alarm for this application has already been set, it is replaced with the new alarm. Action code notifications are sent after the alarm is triggered and can be used by the application to set the next alarm.

Functions for System Use Only

AlmAlarmCallback

Prototype `void  AlmAlarmCallback (void)`

WARNING: This function for use by system software only.

AlmCancelAll

Prototype `void  AlmCancelAll (Boolean enable)`

WARNING: This function for use by system software only.

AlmDisplayAlarm

Prototype `void  AlmDisplayAlarm (Boolean displayOnly)`

WARNING: This function for use by system software only.

AlmEnableNotification

Prototype `void AlmEnableNotificatio(Boolean enable)`

WARNING: This function for use by system software only.

AlmInit

Prototype `Err AlmInit (void)`

WARNING: This function for use by system software only.

Sound Manager Functions

SndDoCmd

Purpose

Send a sound manager command to a specified sound channel.

NOTE: Passing NIL for the channel pointer causes the command to be sent to the shared sound channel.

Prototype

```
Err SndDoCmd (  VoidPtr chanP,
                SndCommandPtr cmdP,
                Boolean noWait)
```

Parameters

-> chanP	Pointer to sound channel. Present implementation doesn't support multiple channels. Must be zero.
-> cmdP	Pointer to a `SndCommandType` structure which contains command parameters.
-> noWait	0 = await completion !0 = immediate return (asynchronous) asynchronous mode is not presently supported

Result

0	No error.
`sndErrBadParam`	Invalid parameter.
`sndErrBadChannel`	Invalid channel pointer.
`sndErrQFull`	Sound queue is full.

SndGetDefaultVolume

Purpose
Return default sound volume levels.

Prototype

```
void SndGetDefaultVolume ( UIntPtr alarmAmpP,
                           UIntPtr sysAmpP,
                           UIntPtr defAmpP)
```

Parameters

<-> alarmAmpP	Pointer to storage for alarm amplitude.
<-> sysAmpP	Pointer to storage for system sound amplitude.
<-> defAmpP	Pointer to storage for master amplitude.

Result
Returns nothing.

Comments
Any pointer arguments may be passed as NULL. In that case, the corresponding setting is not returned.

SndPlaySystemSound

Purpose
Play a standard system sound.

Prototype

```
void  SndPlaySystemSound (SndSysBeepType beepID)
```

Parameters

-> beepID	ID of system sound to play.

Result
Returns nothing.

SndSetDefaultVolume

Purpose
Set the default sound volume levels.

Prototype
```
void SndSetDefaultVolume ( UIntPtr alarmAmpP,
                           UIntPtr sysAmpP,
                           UIntPtr defAmpP)
```

Parameters

-> alarmAmpP	Pointer to alarm amplitude (`0-sndMaxAmp`).
-> sysAmpP	Pointer to system sound amplitude (0-sndMaxAmp).
-> defAmpP	Pointer to master amplitude (`0-sndMaxAmp`).

Result
Returns nothing.

Comments
Any pointer arguments may be passed as NULL. In that case, the corresponding setting are not affected.

Functions for System Use Only

SndInit

Prototype
```
Err SndInit(void)
```

WARNING: This function for use by system software only.

15

Pen, Key, and Graffiti Functions

Pen Manager Functions

PenCalibrate

Purpose	Set the calibration of the pen.

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.
Prototype	`Err PenResetCalibration (void)`
Parameters	None.
Result	Always returns 0.
Comments	Called by Preferences application before capturing points when calibrating the digitizer.
See Also	`PenCalibrate`

WARNING: The digitizer is off after calling this routine and must be calibrated again!!!

Functions for System Use Only

PenClose

Prototype `Err PenClose (void)`

WARNING: This function for use by system software only.

PenGetRawPen

Prototype `Err PenGetRawPen (PointType* penP)`

See Instead `EvtDequeuePenPoint`

WARNING: This function for use by system software only.

PenOpen

Prototype
```
Err PenOpen (void)
```

WARNING: This function for use by system software only.

PenSleep

Prototype
```
Err PenSleep (void)
```

WARNING: This function for use by system software only.

PenRawToScreen

Prototype
```
Err PenRawToScreen (PointType* penP)
```

WARNING: This function for use by system software only.

PenScreenToRaw

Prototype
```
Err PenScreenToRaw (PointType* penP)
```

WARNING: This function for use by system software only.

PenWake

Prototype
```
Err PenWake (void)
```

WARNING: This function for use by system software only.

Key Manager Functions

KeyCurrentState

Purpose
Return bit field with bits set for each key that is currently depressed.

Prototype
`DWord KeyCurrentState (void)`

Parameters
void

Result
DWord 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.

Prototype
```
Err KeyRates (  Boolean set,
                WordPtr initDelayP,
                WordPtr periodP,
                WordPtr doubleTapDelayP,
                BooleanPtr 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	Max 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 does not enqueue keys unless the queue is already empty.

Result Returns 0 if no error.

See Also `KeyCurrentState`

Functions for System Use Only

KeyBootKeys

Prototype `DWord KeyBootKeys (void)`

WARNING: This function for use by system software only.

KeyHandleInterrupt

Prototype
```
ULong KeyHandleInterrupt(  Boolean periodic,
                           DWord status)
```

WARNING: This function for use by system software only.

KeyInit

Prototype
```
Err KeyInit (void)
```

WARNING: This function for use by system software only.

KeyResetDoubleTap

Prototype
```
Err KeyResetDoubleTap (void)
```

WARNING: This function for use by system software only.

KeySleep

Prototype
```
Err KeySleep (   Boolean untilReset,
                 Boolean emergency)
```

WARNING: This function for use by system software only.

KeyWake

Prototype
```
Err KeyWake (void)
```

WARNING: This function for use by system software only.

Graffiti Manager Functions

GrfAddMacro

Purpose
Add a macro to the macro list.

Prototype
```
Err GrfAddMacro (  CharPtr nameP,
                   BytePtr macroDataP,
                   Word 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`

GrfAddPoint

Purpose
Add a point to the Graffiti point buffer.

Prototype
```
Err GrfAddPoint (PointType* ptP)
```

Parameters

ptP	Pointer to point.

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.
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.	
Prototype	`Err GrfDeleteMacro (Word index)`	
Parameters	index	Which macro to delete.
Result	Returns 0 if no error, or `grfErrNoMacros`, `grfErrMacroNotFound` if an error occurs.	
See Also	`GrfAddMacro`	

GrfFindBranch

Purpose	Locate a branch in the Graffiti dictionary by flags.	
Prototype	`Err GrfFindBranch (Word flags)`	
Parameters	flags	Flags of the branch we're searching for.
Result	Returns 0 if no error, or `grfErrNoDictionary` or `grfErrBranchNotFound` if an error occurs.	
See Also	`GrfCleanState`, `GrfInitState`	

GrfFilterPoints

Purpose	Filter the points in the Graffiti point buffer.
Prototype	`Err GrfFilterPoints (void)`
Parameters	None.
Result	Always returns 0.
See Also	`GrfMatch`

GrfFlushPoints

Purpose	Dispose of all points in the Graffiti point buffer.
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.

Prototype

```
Err GrfGetAndExpandMacro(CharPtr nameP,
                         BytePtr macroDataP,
                         WordPtr 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.

Prototype
```
Err GrfGetGlyphMapping( Word glyphID,
                        WordPtr flagsP,
                        void* dataPtrP,
                        WordPtr dataLenP,
                        WordPtr uncertainLenP)
```

Parameters

glyphID	Glyph ID to lookup.
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.

Prototype
```
Err GrfGetMacro( CharPtr nameP, BytePtr macroDataP,
                 WordPtr 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.
Prototype	`Err GrfGetMacroName (Word index, CharPtr 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.
Prototype	`Err GrfGetNumPoints (WordPtr 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.
Prototype	`Err GrfGetPoint (Word index, PointType* pointP)`
Parameters	index — Which point to get. pointP — Returned point.
Result	Returns 0 if no error, or `grfErrBadParam` if an error occurs.
See Also	`GrfAddPoint`, `GrfGetNumPoints`

GrfGetState

Purpose Returns the current shift state of Graffiti.

Prototype
```
Err GrfGetState(   Boolean* capsLockP,
                   Boolean* numLockP,
                   WordPtr 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.

Result Always returns 0.

See Also `GrfSetState`

GrfInitState

Purpose Reinitialize the Graffiti dictionary state.

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.

Prototype:

```
Err GrfMatch (   WordPtr flagsP,
                 void* dataPtrP,
                 WordPtr dataLenP,
                 WordPtr uncertainLenP,
                 GrfMatchInfoPtr matchInfoP)
```

Parameters:

flagsP	Glyph flags are returned here.
dataPtrP	Return text is placed here.
dataLenP	Size of dataptr on exit; number of characters returned on exit.
uncertainLenP	Return number of uncertain characters.
matchInfoP	Array of `grfMaxMatches`, or nil.

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.

Prototype

```
Err GrfMatchGlyph (  GrfMatchInfoPtr matchInfoP,
                     Word maxUnCertainty,
                     Word 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.

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: 0 if recognized.

Comments: Called by `SysHandleEvent` when a pen-up 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.

Prototype

```
Err GrfSetState( Boolean capsLock,
                 Boolean numLock,
                 Boolean upperShift)
```

Parameters

capsLock	Set to TRUE to turn on caps 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`

SysShortCutListDialog

Purpose Pop up the Graffiti ShortCut list as a field object with the focus.

Prototype

```
void SysGrfShortCutListDialog (void)
```

Parameters

event	Pointer to an `EventType` structure.

Result The field's text chunk is changed.

See Also `GrfGetMacro`, `GrfGetMacroName`

Functions for System Use Only

GrfFieldChange

Prototype

```
Err GrfFieldChange(  Boolean resetState,
                     UIntPtr characterToDelete)
```

WARNING: System Use Only.

GrfFree

Prototype	`Err GrfFree(void)`

WARNING: System Use Only.

GraffitiShift Functions

GsiEnable

Purpose	Enable or disable the Graffiti-shift state indicator.
Prototype	`void GsiEnable (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.

GsiEnabled

Purpose	Return TRUE if the Graffiti-shift state indicator is enabled, or FALSE if it's disabled.
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.
Prototype	`void GsiInitialize (void)`
Parameters	None.
Result	Returns nothing.

GsiSetLocation

Purpose	Set the display-relative position of the Graffiti-shift state indicator.
Prototype	`void GsiSetLocation (short x, short y)`
Parameters	x, y Coordinate of left side and top of the indicator.
Result	Returns nothing.
Comments	The indicator is not redrawn by this routine.

GsiSetShiftState

Purpose Set the Graffiti-shift state indicator.

Prototype

```
void GsiSetShiftState ( Word lockFlags,
                        Word tempShift)
```

Parameters

lockFlags	`glfCapsLock` or `glfNumLock`.
tempShift	The current temporary shift.

Result Returns nothing.

Comment This function affects only the state of the UI element, not the underlying Graffiti engine.

See Also `GrfSetState`