Welcome to the

Palm OS Cookbook

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®

Palm OS™ Cookbook

Some information in this manual may be out of date.
Read all Release Notes files for the latest information.

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

1

Building a Palm OS Application

Building an application for the Palm OS device involves a few simple steps, discussed in this chapter:

- Checking the File Hierarchy
- Creating a Makefile
- Compiling the Application
- Loading the Application Onto the Device

When you're working with your application on the Macintosh using the Pilot Simulator, the CodeWarrior project file contains the information needed to run your application. When you build the application for the device, a Makefile containing similar information is used.

NOTE: Phase 20 of the Tutorial steps you through the process of building an application and running it on the Palm OS device. The example code provides a working Makefile and application file hierarchy.

Checking the File Hierarchy

To build the executable for your application, you need to have several files set up in the correct folder hierarchy. The Makefile uses these files when you build the executable, as discussed in the following sections.

The following files are part of the set-up. *Name* is the name of the application; *App* is the folder in which the application resides. All files and folders need to be located within the Palm OS SDK tree:

Filename	**File**	**Folder**
Name.c	Source code; can be more than one file.	*App*:Src
Name.h	Header file; can be more than one file.	*App*:Src
Name.r	List of resources used when building for the device.	*App*:Src
*Name*Rsc.c	List of resources used when building for the Simulator.	*App*:Src
*Name*Rsc.h	Header file containing resource ID definitions.	*App*:Src
*.rsrc	Resource file, usually one file per form.	*App*:Rsc
Makefile		*App*
Name.µ	Code Warrior Project file.	*App*

Creating a Makefile

The Makefile tells the compiler which source and resource files belong to the application and how to compile it. You can use the sample Makefile that is included in Phase 20 of the tutorial in your application, but you need to make the following changes:

- In the Compile Options section, set `ERROR_CHECK_LEVEL` as desired, choosing from these options:
  - 0—ERROR_CHECK_NONE (no error checking)
  - 1—ERROR_CHECK_PARTIAL (display fatal errors only)
  - 2—ERROR_CHECK_FULL (display fatal or nonfatal errors)

  For example, to set error checking to full, the Makefile should say:

  ```
  -d ERROR_CHECK_LEVEL=2 ∂
  ```

  For more information, see “The Error Manager,” in Chapter 6, “Palm OS Managers” of “Developing Palm OS Applications, Part I.”

- If you're compiling a localized version, set the COUNTRY and LANGUAGE compiler options. Currently, you have these choices:
  - 0—COUNTRY_UNITED_STATES
  - 1—COUNTRY_FRANCE
  - 2—COUNTRY_GERMANY
  - 0—LANGUAGE_ENGLISH
  - 1—LANGUAGE_FRENCH
  - 2—LANGUAGE_GERMAN

  For example, to set the country to France and the language to French, the Makefile should contain these lines:

  ```
  -d COUNTRY=1∂
  -d LANGUAGE=1∂
  ```

  If you compile with a non-zero COUNTRY or LANGUAGE option, the compiler uses the resources in the folder that has a name corresponding to the options.
- In the Object List section, list each source file (.c file) except the *Name*.Rsc.c file.
- In the Objects section, list each resulting object (.o) file.
- In the Compiles section, list each source file and corresponding object file.
- In the Final Link section, provide the name of the target. Set the creator identifier assigned to the application executable by Palm Developer Support. It should match the `appFileCreator` defined in the *Name*.h file. See the example file for more information.

Listing 1.1 shows the Makefile used in Phase 20 of the tutorial.

Listing 1.1 Sample Makefile

```
###################################################
#   Makefile for Pilot MemoPad Tutorial
#   Target:     MemoPad
#
#   Compiles and links the MemoPad tutorial
#   application for running on the Palm OS device.
#
```

```
#   To run this make, use the MPW Directory Menu
#   to set the current directory to the MemoPad
#   directory, go to the Build menu, select
#  "Build" and enter "MemoPad" into the dialog box.
#
###################################################

###################################################
# Set up paths
###################################################
LIB_DIR = :::Libraries:PalmOS:
INC_DIR = :::Incs:
SRC_DIR = :Src:
OBJ_DIR = :Obj:


###################################################
# Set up Compiler
###################################################
# Use Metrowerks compiler
CC = MWC68K
CPP = MWC68K
LINK = MWLink68K


###################################################
# Compile Options
#
# The most likely options you might change are
# COUNTRY, LANGUAGE, and ERROR_CHECK_LEVEL.
#
###################################################
C_OPTIONS = ∂
        -d COUNTRY=0∂
        -d LANGUAGE=0∂
        -d ERROR_CHECK_LEVEL=2 ∂
        -d CMD_LINE_BUILD ∂
        -d EMULATION_LEVEL=0 ∂
        -d ENVIRONMENT=0 ∂
        -d MEMORY_FORCE_LOCK=1 ∂
        -nosyspath ∂
        -i ":Src:" ∂
        -i "{INC_DIR}" ∂
        -i "{INC_DIR}System:" ∂
        -i "{INC_DIR}UI:" ∂
        -i "{INC_DIR}Hardware:" ∂
        -model near ∂
        -intsize 2 ∂
        -maxerrors 3 ∂
        -opt speed -opt global -opt peep ∂
        -mbg on ∂
        -b ∂
        -d PILOT_PRECOMPILED_HEADERS_OFF


LINK_OPTIONS = -single -custom


###################################################
# Object List
#
# Each of the source files must be listed here.
###################################################
OBJECTS = ∂
       "{LIB_DIR}StartupCode.c.o" ∂
       "{OBJ_DIR}MemoPad.c.o"



###################################################
# Compiles
#
# There should be one compile statement per source
# file. On the right side of the first line is a
# list of files the target depends on. The target
# is only regenerated if one of those files changes
```

```
###################################################
"{OBJ_DIR}MemoPad.c.o" ƒ MakeFile
"{SRC_DIR}MemoPad.c"
   {CPP}  -o "{OBJ_DIR}MemoPad.c.o" ∂
      "{SRC_DIR}MemoPad.c" ∂
      {C_OPTIONS}


###################################################
# Final Link
##################################################
MemoPad ƒƒ MakeFile {OBJECTS} "{SRC_DIR}MemoPad.r"
#
# Link all of the code together.
#
  {LINK} {LINK_OPTIONS} -t rsrc -c RSED ∂
    {OBJECTS} ∂
    -o MemoPad.code
  Delete -i  "MemoPad"
#
# Build a description of all the resources used by
# the application
#
  {CC} -d RESOURCE_COMPILER ∂
      {C_OPTIONS} ∂
      -e ∂
      "{SRC_DIR}MemoPad.r" > MemoPad.i
#
# Process the ui resources and combine them and
# the code into a single file for use on the
# device. Optionally set other data about the
# device file like its type, creator, version
# number, name, etc. Type PilotRez in the MPW
# Worksheet to see a description of PilotRez
# options.
#
  PilotRez -v 1 -t appl -c MEMP -it MemoPad.i -ot
"MemoPad"
```

```
#
# Pilot Desktop prefers device files to end with
# .prc
#
  Duplicate -y "MemoPad" "MemoPad.prc"
#
# Pilot Debugger is used by application developers
# to send the device application to the device.
# Place a duplicate of the file in a place handy
# for the debugger.
#
  Duplicate -y "MemoPad.prc"  "{PilotDir}"Debugger
```

Compiling the Application

With both the Makefile and the resource file for your application complete, launch MPW, then follow these steps:

1. From the Directory menu, choose Set Directory to set the directory to your application's directory, that is, the directory that contains the Rsc and Src folders and the Makefile (called *App* in the listing above).
2. From the Build menu, choose Full Build to build the executable for your application using the newly created Makefile.
3. When prompted, enter the target name defined in the Final Link section of the Makefile (for example, MemoPad).

   A successful build using the example Makefile copies the executable to the Debugger folder.

Loading the Application Onto the Device

The current procedure for downloading an application onto a Palm OS device involves using the Pilot Debugger of the Palm OS. You first establish a connection between the Macintosh and the Palm OS device, then download the executable.

To load an application onto the device, follow these steps:

1. Launch the PilotDebugger application found in the Debugger folder of the Palm OS SDK on the Macintosh.
2. Click on the Console window to bring it to the front.
3. Place a Palm OS device in a cradle connected to the Macintosh serial port.
4. On the Palm OS device, launch the Preferences application.
5. On the Palm OS device, use Graffiti to enter the shortcut ".2." The sequence consist of four strokes: the shortcut stroke, followed by two taps (to generate a period), followed by the stroke for the number 2. Remember to enter the number stroke in the right side of the Graffiti area.

   This sequence tells the device to listen to the Debugger.
6. If you've successfully entered the ".2" shortcut sequence, **and** the Palm OS device is connected to the Macintosh correctly through the serial port, **and** the Pilot Debugger application is running with the Console window in front, **then** you'll see the text "Ready..." show up on the Console window on the Macintosh. If sound is enabled, the device clicks.
7. To verify that the Palm OS device is responding to commands from the Macintosh, type into the Console window

   ```
   dir 0
   ```

   then press the Enter (not Return) key or Cmd-Return.

   A list of databases should appear in the Console window. If something is wrong, you'll notice a few seconds delay and then see a message like `### Error $00000404 occurred`.

   Note that if the Console window is not the active window when you enter the ".2" shortcut, you'll miss the "Ready..." message but will still be able to enter commands in the Console window.

   Entering the ".2" shortcut sequence a second time does **not** print another "Ready..." message. If you're not sure whether the connection is established, enter `dir 0` in the Console window and see if there is a response.

You can now load your application onto the device; a copy of the executable was placed in the Debugger folder when you compiled the application with the Makefile.

8. In the Pilot Debugger Console window, type

   ```
   import 0 application
   ```

   then press Enter (not Return).

The “0” in the import command sends the application to the first memory card. *`Executable`* is the name of the executable to send. The Debugger looks for the executable in the Debugger folder. The executable name is not case sensitive here, but you must include the extension, if there is one (usually .prc).

When the import is complete, the launcher screen on the Palm OS device displays the application you just downloaded and you can select it to run it.

When you finish using the Pilot Debugger, you have to reset the Palm OS device. (The reset hole is on the back of the device.) Until you have done that, the serial port of the Palm OS device is held by the remote debug nub, waiting for more commands from the Pilot Debugger. This means that HotSync fails and power is continually fed to the serial port from the batteries.

When you send other versions of an application to the Palm OS device, an existing version must be removed first. The Debugger command `del 0` *`databasename`* accomplishes the removal, where *`databasename`* is the name of the database shown in the directory listing (`dir 0`). In this case, the name is case sensitive. Note that *`databasename`* refers to either an executable or an application’s database.

2

Using the Pilot Simulator

The Pilot Simulator lets you run and test your application by providing a graphic representation of the Palm OS device within the Macintosh environment.

To test your application, you can click the mouse on the representation of the device's physical controls (including the silk-screened icons) or click any of the menus, buttons, or other user interface items your application provides. You can also use the mouse to write in the Graffiti area of the representation of the device that's visible on the Macintosh. As you do this, you can trace events or interact with your application from the Console window.

This chapter serves as a reference to the Simulator, discussing these topics:

- Simulator Menu Commands
- Simulator Console Commands
- Differences Between Pilot Simulator and Palm OS Hardware
- Debugging Memory Management

Simulator Menu Commands

This section discusses all Simulator menus (from left to right) and describes each command (from top to bottom).

The following menus are discussed:

- The File Menu
- The Edit Menu
- The Window Menu
- The Replay Menu
- The Gremlin Menu

The File Menu

Save Card 0...

Writes the contents of memory card 0 to a file called Pilot Card 0 in the currently simulated application's folder.

The card corresponds to the memory card on the actual Palm OS device, on which all application databases and add-on applications are stored.

Save Card 1...

Writes the contents of memory card 1 to a file called Pilot Card 1 in the currently simulated application's folder.

Currently, it makes sense only to save card 0.

Save Before Quitting

Saves a snapshot of the contents of both memory cards after `StopApplication` has been called.

When an application exits, it saves certain information like preferences to the memory card. If you use this command, the Simulator saves what the application would save to the memory card to a file called Pilot Card 0.

Quit

Quits the application.

The Edit Menu

The Edit menu offers the standard Macintosh editing commands for the Console window and the Event Trace window.

The Window Menu

The Window menu provides access to two special windows: the Console window and the Event Trace window. Close either window by clicking the button in the top-left corner or by deselecting that window in the menu.

Console

Activates the Console window. Type Help and press Enter (or Cmd-Return) in the Console window for a list of available commands.

See Simulator Console Commands for a list of all commands.

Event Trace

Displays the Event Trace window. The Event Trace window displays the last 100 events generated by the system software and application.

Chapter 4, “Palm OS Events,” of “Developing Palm OS Applications, Part I” lists and discusses all Palm OS events.

The Replay Menu

The Replay menu allows you to record pen and key events to a script file. You can then use the script file to replay the same events. This is useful for testing and repeating problem cases. The general process of using the commands in this menu is as follows:

1. Select the Record command.
2. Record pen and key events.
3. Deselect the Record command.
4. Select the Replay command. By default, the last script saved is already displayed in the file selection dialog.
5. Choose Pause and Step during replay to look in detail at the events that are executed.

Record

Begins recording pen and key events to a file. To stop recording, deselect this command.

Break

Inserts a stop into the script so it stops during replay. Does not stop the recording process.

Save As

Saves the recorded script to a file you specify. By default, the Simulator saves a script to a file with the extension .LOG whenever you stop recording. Use the Save As command to create an additional copy of that script file. The default filename is "Pilot Script."

Playback

Plays back a previously recorded script. You are prompted to locate the script.

Pause

Pauses playback of a script. This command is available during playback, but not during the recording process.

Step

Plays back the next pen or key event, then pauses. This command is available during playback, but not during the recording process.

Realtime

Tries to execute the script at the rate at which it was recorded. With this option off, scripts execute as fast as possible. Realtime is useful when timing-dependent UI elements such as repeating buttons are used.

The Gremlin Menu

Gremlins are a facility to generate random pen and key events. You can use the facility to reveal program problems. Each Gremlin is a unique sequence of random taps, strokes, and so on. Red lines indicate how the pen was moved over the screen by the Gremlin.

Gremlins range from 0 to 1000 and each Gremlin is repeatable. From the menu, you can invoke only Gremlin 0. To start up any other Gremlin, use the Console window (see the Gremlins command under Simulator Commands).

New

Runs Gremlin number 0. Iterates through all events in that Gremlin, running continuously.

Step

Performs the next Gremlin event, then stops.

Resume

Resumes running continuously after a step or stop.

Stop

Stops generating Gremlin events.

Simulator Console Commands

This section provides descriptions and syntax for the commands available in the console window. It covers the following commands:

- System Commands
- Card Info Commands
- Heap Utility Commands
- Chunk Utilities
- Database Utilities
- Miscellaneous Utilities
- Record Utilities
- Resource Utilities
- Debugging Utilities
- Simulator Commands

NOTE: Console commands are not case sensitive!

System Commands

Command	Result	Syntax
Feature	Display, get, register, or unregister feature(s).	feature [options...] -all: Display a list of all known features -unreg <creator> <num>: Unregister feature (use only on RAM features) -get <creator> <num> : Get a feature -set <creator> <num> <value>: Set a feature
Kinfo	Get kernel info.	kinfo [options...] -sem <id>|all : get semaphore info -tmr <id>|all : get timer info

Card Info Commands

Command	Result	Syntax
cardformat	Format a memory card.	cardformat <cardNo> <cardName> <manufName> <ramStoreName> cardNo, manufName, and ramStoreName are strings. WARNING: The Simulator’s behavior is unpredictable if you reformat the card containing your application.
cardinfo	Get info on a memory card.	cardinfo <cardNo>
storeinfo	Get info on a memory store.	storeinfo <cardNo>

Heap Utility Commands

Command	Result	Syntax
HL	List all the heaps on a memory card.	hl <cardNo>
HI	Initialize a memory heap.	hi <hex heapID>
HD	Do a heap dump.	hd <hex heapID>
HT	Do a heap total.	ht <hex heapID>
HC	Compact a memory heap.	hc <hex heapID>
HChk	Do a heap check.	hchk <hex heapID>
HS	Do a heap scramble.	hs <hex heapID>
HF	Allocate all free bytes, except specified number of bytes.	hf <hex heapID> [free bytes]

NOTE: For more information, see Debugging Memory Management.

Chunk Utilities

Cmd	Result	Syntax
New	Allocate a new chunk in a heap.	new <hex heapID> <hex chunkSize> [options...] -n : non-movable -c : fill contents -o <ownerID> : set owner (0-14) -near <ptr> : In same heap as <ptr> so <heapID> is ignored -lock : pre-lock
Free	Dispose of a chunk.	free <hex chunk ptr/ID>[a] [options..] -card <cardNo> : card number when local ID specified instead of chunk ptr
Lock	Lock a chunk.	lock <hex chunk ptr/ID>[a] [options..] -card <cardNo> : card number when local ID specified instead of chunk ptr
Unlock	Unlock a chunk.	unlock <hex chunk ptr/ID>[a] [options..] -card <cardNo> : card number when local ID specified instead of chunk ptr
Info	Get info on a chunk.	info <hex chunk ptr/ID>[a] [options..] -card <cardNo> : card number when local ID specified instead of chunk ptr
Resize	Resize an existing chunk.	resize <hex chunk ptr/ID>[a] <hex newSize> [options..] -c : check and fill contents -card <cardNo> : card number when local ID specified instead of chunk ptr
Set-Owner	Set the owner of a chunk.	setowner <hex chunk ptr/ID> <owner> [options..] -card <cardNo> : card number when local ID specified instead of chunk ptr

a. The hex chunk ptr/ID is the value in the Start column of a heap dump (HD).

Database Utilities

Cmnd	Result	Syntax
Import	Import a data-	import [-r] <cardNo> <filename>
	base.	-r : Import from Macintosh resource file
		-u : Create new unique IDs
		-d : Update creation and modification dates
Export	Export a data-	export [-m] <cardNo> <filename>
	base.	-m : Export as Macintosh resource file
Create	Create a data-	create <cardNo> <name> [options...]
	base.	-t <type> : Database type, 4 characters
		-c <creator> : Database creator, 4 characters
		-v <version> : Database version
		-r : Resource database
Del	Delete a data-	del <cardNo> <databaseName>
	base.	
Dir	List database	dir <cardNo> | <searchOptions> [displayOptions]
	directory.	<searchOptions>:
		-t search by type
		-c search by creator
		-latest search for only latest version of each database
		<displayOptions>
		-a show all info for each database
		-n show name
		-id show ChunkID
		-s show size
		-r show number of records
		-at show attributes
		-v show version
		-d show dates
		-m show modification
		-i show info fields
		-tc show type & creator
		Note: follow any option with - to turn it off.

Cmnd	Result	Syntax
Open	Open a data-base.	open <cardNo> <name> [options...] -r open read-only -p leave open
Close	Close a data-base.	close <access ptr>
Opened	List all open databases.	opened
SetInfo	Set info in a database.	setinfo <cardNo> <dbName> [options...] <options>: -v <version> set version -m <modification #> set modification number -n <name> set name

Miscellaneous Utilities

Command	Result	Syntax
SimSync	Simulate a Sync on a database. Deleted records are removed and dirty flags are cleared.	simsync [access ptr][a]
SysAlarmDump	Display the alarm table	sysalarmdump

a.Use the opened command to find the access ptr for a database.

Record Utilities

Command	Result	Syntax
AttachRecord	Attach a record to a database.	attachrecord <access ptr>[a] <record handle> <index> [options...] -r replace existing record
DetachRecord	Detach a record from a database.	detachrecord <access ptr>[a] <index>
AddRecord	Add a record to a database.	addrecord <access ptr>[a] <index> <record text>
DelRecord	Delete a record from a database.	delrecord <access ptr>[a] <index>
Change-Record	Replace record in a database.	changerecord <access ptr>[a] <index> <record text>
ListRecords	List records in a database.	listrecords <access ptr>[a]
SetRecordInfo	Set info on a database record.	setrecordinfo <access ptr>[a] index [options...] -a <hex attr> set attributes -u <uniqueID> set unique ID
MoveRecord	Move a record from one index to another.	moverecord <access ptr>[a] <from> <to>
FindRecord	Find a record by unique ID.	findrecord <access ptr>[a] <id>

a.Use the opened command to find the access ptr for a database.

Resource Utilities

Command	Result	Syntax
GetResource	Get a resource.	getresource -t <type> -id <id>
ListResources	List resources in a database.	list resources <access ptr>
SetResourceInfo	Set resource info.	setresourceinfo <access ptr> <index> [options...] -t <resType> set resource type -id <resID> set resource id
AddResource	Add a resource to a database.	addresource <access ptr> -t <type> -id <id> <record text>
DelResource	Delete a resource from a database.	delresource <access ptr> <index>
ChangeResource	Change a resource in a database.	changeresource <access ptr> <index> <resource text>

Debugging Utilities

Command	Result	Syntax
DM	Display memory.	dm <addr> [<count>]
SB	Set byte	sb <addr> <value>
MDebug	Set memory manager debug mode. Sets the level of memory checking used whenever a memory manager API function is executed. This command is helpful in tracking memory corruption bugs.	mdebug [options..] Shortcuts: -full : Full checking (slowest) -partial : Partial checking (faster) -off : No checking (fastest) Fine tuning (Which heaps are checked/scrambled): -a : check/scramble **all** heaps each time -a- : check/scramble affected heap only Heap checking: -c : check heap(s) on some Mem calls -ca : check heap(s) on every Mem call -c- : turn off heap checking Heap scrambling: -s : scramble heap(s) on some Mem calls -sa : scramble heap(s) every Mem call -s- : turn off heap scramble Free chunk checking: -f : check free chunk contents -f- : don't check free chunk contents Minimum dynamic heap free space recording: (recorded in the global `GMemMinDynHeapFree`) -min : record minimum free space in dynamic heap -min- : don't record minimum free space

Simulator Commands

Command	Result	Syntax
Gremlin	Activate a particular Gremlin until a certain event.	gremlin [num] [until] num—gremlin number until—number of input events to generate.
PopulateDB		PopulateDB <number of records> <max. record size>[15]
Date	Display or set the current date.	date [mm/dd/yyyy]
Time	Display or set the current time in 24-hour format; seconds are not displayed.	time [hh:mm[:ss]]
AlarmSound	Display or set the alarm sound state.	alarmsound [on | off]
SystemSound	Display or set the system sound state.	systemsound [on | off]
SyncNotify	Send HotSync notification to the current application.	syncnotify

Differences Between Pilot Simulator and Palm OS Hardware

There are a few differences between an application running under the Pilot Simulator and one running on a Palm OS device, which can cause difficulties during debugging. In particular, the Simulator allows an application to do a few things that won't work on the device. If your application runs under the Simulator but doesn't run on the device, check for these potential problems:

- Does your application use standard C run-time library calls? These calls work under the Simulator but may not work on the device. The calls include memory management routines (malloc, free), string operations (strcpy, strcmp), mathematical functions (rand, cos), and so on. There are Palm OS equivalents for most of these functions. Note that even if a standard C run-time library call works on the device, it unnecessarily enlarges your application.
- Are you writing to storage RAM without using `DMWrite`? The Palm OS device enforces write protection and the Simulator does not. On the device, a bus error results. Under the Simulator, there are no problems.
- Are you accessing 16- or 32-bit memory values at odd addresses? The Simulator allows this, but the device generates a bus error. This often happens when you are working with packed data rather than defined structures.
- Does your application attempt intra-application jumps of more than 32K? This results in link errors. You can often overcome this problem by rearranging your code.
- Does your application overflow the stack? Applications running on the device have only a 2K stack. Applications running under the Simulator have a much larger stack. If your application stores a lot of data in local variables, this problem could arise. Store the data in global variables or in allocated database chunks instead.
- Pointer errors have a much more dramatic effect on the device than under the Simulator. This is due to the greater density of data in the device's memory than in Macintosh memory. Any bad pointer values, array overwrites, and so on, are much more likely to destroy important data on the Palm OS device than on the Macintosh.

In addition, there are a few harmless differences between the Palm OS device and the Simulator:

- The buttons on the front of the simulated device do not switch applications because you can run only one application at a time under the Simulator.
- The four buttons are simulated by the F9, F10, F11, and F12 keys on your keyboard.
- The scroll up and scroll down buttons are simulated by the page up and page down keys on your keyboard.

Debugging Memory Management

This section looks in some detail at debugging memory management. Using commands that can be executed from the Console window of the Simulator or from the Console window of the Pilot Debugger, you can make sure that your application manages memory correctly. This includes making sure it is not corrupting memory, causing unnecessary heap fragmentation, or leaving unused memory chunks around (that is, causing memory leaks).

This section assumes that you understand how memory is structured in the Palm OS environment. In particular, you need to know how memory is subdivided into heaps, the different types of memory chunks (movable and nonmovable) in heaps, and how the data manager stores information. “Palm OS Memory Management” in “Developing Palm OS Applications” describes this in detail.

The Console window accepts the same memory debugging commands in either the Pilot Simulator or the Pilot Debugger. The only difference is that when you run the Pilot Debugger, each command you enter in the Console window communicates with the device over the serial port in order to carry out its function on the device.

Two classes of memory management commands are available. These are discussed in the next two sections:

- Displaying Memory Information
- Manipulating Memory (commands that actively manipulate memory or put the memory manager into different modes)

Displaying Memory Information

A number of commands are available for displaying how memory is allocated. At the highest level, there are commands for displaying the available heaps and summary information about each heap. At the lowest level, there are commands for displaying how each heap is subdivided and for getting information on each chunk in a heap.

Card Info

Syntax `CardInfo <cardno>`

Description Displays the total amount of RAM and ROM on a memory card. Also displays the total amount of free space in RAM and the total number of heaps. This is especially helpful when you want a snapshot overview of the memory situation. The current Palm OS device only has one memory card slot, so the <cardno> argument has to be 0 (for the first memory card).

Example

```
cardinfo 0
Name: Card 0
Manuf: Palm Computing
Version: 0001
CreationDate: 12345678
ROM Size: 00000000
RAM Size: 0003811A
Free Bytes: 000315F6
Number of heaps: #4
```

Heap List

Syntax `HL <cardno>`

Description Displays a list of heaps on a memory card. For each heap, shows the heapID, pointer, size, free bytes, biggest free chunk, and flags. The heapID is important because it must be passed to the commands which display more detailed information about each heap.

Example

```
hl 0
```

Heap Total

Syntax `HT <heapID>`

Description Displays summary information about a heap. This includes the total heap size; the number of master pointers available for movable chunks (`numHandles`); and the total number of free chunks, movable chunks, and nonmovable chunks currently allocated.

Example

```
ht 1
Displaying Heap ID: 0001, mapped to 00B2B20A
---------------------------------------------------
Heap Summary:
  flags:              0000
  size:               10000
  numHandles:         #200
  Free Chunks:        #1      (FC42 bytes)
  Movable Chunks:     #3      (0094 bytes)
  Non-Movable Chunks: #0      (0000 bytes)
```

Heap Dump

Syntax `HD <heapID>`

Description Displays all the chunks in a heap. For each chunk, it shows the start address, the handle if it's a movable chunk, the local ID, the requested size, the actual size, the lock count, the owner ID, the flags and some identifying information about the chunk, including the database it belongs to, its resource type, and ID or record number.

If you dump the dynamic heap (heapID 0), the identifying information indicates whether or not the chunk is allocated by one of the system managers.

Example

```
hd 1
```

Heap Check

Syntax `HChk <heapID>`

Description Checks the integrity of a heap and displays an error message if it detects an error in the heap structure. The `HD` (Heap Dump) and `HT` (Heap Total) commands also check the integrity but sometimes `HChk` is more convenient because it doesn't print extra information.

Example

```
hchk 0
Heap OK
```

Chunk Info

Syntax `Info -card <cardNo> <localID>`

Description Most useful for converting a Local ID to a chunk pointer or handle. Given a card number and Local ID (in hex), returns information about the chunk that the Local ID refers to. This includes a pointer to the start of the chunk data, the handle of the chunk (if it's a movable chunk), the flags, size, owner ID, lock count, and the heap ID of the heap that the chunk resides in.

Example

```
info -card 0 8123
Info on Chunk at: 00B2B56C
Handle: 80B2B212
Flags: 0000
Size: 005A
Owner: #1
LockCount: #0
Heap ID: 0001
```

Manipulating Memory

This section describes the commands available for manipulating memory. For example, there are commands for filling a heap, compacting a heap, and scrambling a heap. These are all useful when you want to test the behavior of your application in various memory situations.

Another command, `MDebug`, puts the memory manager into different modes that are helpful for tracking down intermittent memory problems.

Heap Compact

Syntax

```
HC <heapID>
```

Description

Forces a heap to be compacted. This essentially merges all the free chunks together. Normally, the memory manager compacts a heap only when a memory allocation fails.

Example

```
hc 0
Heap Compacted
```

Heap Fill

Syntax

```
HF <heapID> [freeBytes]
```

Description

Fills a memory heap with memory chunks until there are only <freeBytes> free bytes. This is useful for testing the behavior of your application in low-memory situations. If <freeBytes> is not specified, the entire heap is filled, with 0 free bytes remaining.

Example

```
hf 1 100
```

Heap Scramble

Syntax:

```
HS <heapID>
```

Description

Scrambles a heap. As a result, any movable chunks in the heap are moved to another location in the heap. This is useful for detecting situations where your application may have kept around a pointer to an unlocked chunk. After a scramble, any pointers to unlocked chunks are invalid and will most likely cause your application to crash.

Example

```
hs 1
Heap Scrambled
```

Memory Manager Debug Mode

Syntax:

```
MDebug [options..]
  Shortcuts:
    -full     : Full checking (slowest)
    -partial  : Partial checking (faster)
    -off      : No checking (fastest)
  Fine Tuning:
  Which heaps are checked/scrambled:
    -a      : check/scramble ALL heaps each time
    -a-     : check/scramble affected heap only
  Heap Checking:
    -c      : check heap(s) on some Mem calls
    -ca     : check heap(s) on every Mem call
    -c-     : turn off heap checking
  Heap Scrambling:
    -s      : scramble heap(s) on some Mem calls
    -sa     : scramble heap(s) every Mem call
    -s-     : turn off heap scramble
  Free Chunk Checking:
    -f      : check free chunk contents
    -f-     : don't check free chunk contents
```

```
Min Dynamic Heap free space recording:
(Recorded in the global GMemMinDynHeapFree)
  -min    : record minimum free space in
            dynamic heap
  -min- : don't record minimum free space
```

Description

Puts the memory manager into the selected debug mode. Can put it into modes where heaps are automatically scrambled and checked after your application makes certain memory manager calls. Once you put the memory manager into one of these debug modes, you are dropped into the Debugger as soon as a corrupted heap is detected.

Most of the time, the `MDebug` command is used with the `-partial` argument. This puts the memory manager into a mode where on every memory manager call, it scrambles and checks the heap that memory manager call operates on. This makes your application run significantly slower, but the performance degradation is tolerable under the Pilot Simulator running on a Macintosh Quadra or faster.

A more stringent mode is entered with the `-full` argument. This makes the memory manager check every heap on every memory manager call. The performance degradation from this command is usually quite bad, so it's best not to use it until the last stages of verification.

The other options (`-a, -c, -s, -f`) can be used to fine tune the debug mode. You'll notice that `-partial` and `-full` are merely shortcuts that set the these other options for you. After setting the mode using `-partial` or `-full`, you can further fine tune it by entering one or more of these other options.

The last option, `-min`, makes the memory manager record the minimum amount of free space ever detected in the dynamic heap. After setting this option, you can run your application for awhile and then enter the `MDebug` command with no options, to display the minimum amount of free space ever detected in the dynamic heap. This is helpful in determining the "breathing room" in the dynamic heap.

Examples:

```
mdebug -partial
Current mode = 001A
  Only Affected heap checked/scrambled per call
  Heap(s) checked on EVERY Mem call
  Heap(s) scrambled on EVERY Mem call
  Free chunk contents filled & checked

  Minimum dynamic heap free space recording OFF

mdebug -full
Current mode = 003A
  Every heap checked/scrambled per call
  Heap(s) checked on EVERY Mem call
  Heap(s) scrambled on EVERY Mem call
  Free chunk contents filled & checked

  Minimum dynamic heap free space recording OFF

mdebug -min
Current mode = 007A
  Every heap checked/scrambled per call
  Heap(s) checked on EVERY Mem call
  Heap(s) scrambled on EVERY Mem call
  Free chunk contents filled & checked

  Minimum dynamic heap free space recording on:
  Current value: 0 bytes


mdebug
Current mode = 007A
  Every heap checked/scrambled per call
  Heap(s) checked on EVERY Mem call
  Heap(s) scrambled on EVERY Mem call
  Free chunk contents filled & checked

  Minimum dynamic heap free space recording on:
  Current value: 15918 bytes
```

```
mdebug -a-
Current mode = 005A
  Only Affected heap checked/scrambled per call
  Heap(s) checked on EVERY Mem call
  Heap(s) scrambled on EVERY Mem call
  Free chunk contents filled & checked

  Minimum dynamic heap free space recording on:
  Current value: 15918 bytes
```

3

File Transfer With the Pilot Debugger

The Pilot Debugger is a Macintosh application that communicates through the Macintosh serial port with a Palm OS device. It allows you to download applications to the device and to upload data files created on the device to the Macintosh.

This chapter describes how to use the Pilot Debugger for file transfer between a Palm OS device and a Macintosh, discussing these topics:

NOTE: You can also use the Pilot Debugger to debug memory management. See Debugging Memory Management.

Setting Up the Pilot Debugger

To use the Pilot Debugger, you need to connect the Palm OS device, through its cradle, to the Macintosh modem port. For best results, use a high-speed Macintosh modem cable that has a DIN-8 connector on one side and a DB-25 connector on the other. Make sure the cable is marked “high speed”; only those cables have the necessary hardware handshaking lines. Connect the DB-25 end into the Palm OS device cradle using a DB-9 to DB-25 adaptor.

Starting Up the Pilot Debugger

Before the Pilot Debugger can communicate with the Palm OS device, you must launch a background task on the device which watches for commands sent from the Macintosh. To do this, follow these steps:

1. On the Macintosh, launch the Pilot Debugger application found in the Debugger folder of the Palm OS SDK.
2. Click on the Console window to bring it to the front.
3. Place a Palm OS device in a cradle connected to the Macintosh serial port.
4. On the Palm OS device, use Graffiti to enter the shortcut ".2." The sequence consists of four strokes: the shortcut stroke, followed by two taps (to generate a period), followed by the stroke for the number 2. Remember to enter number the stroke on the right side of the Graffiti area.

   This sequence tells the device to listen to the Debugger.
5. If you've successfully entered the ".2" shortcut sequence, **and** the Palm OS device is connected to the Macintosh correctly through the serial port, **and** the Pilot debugger application is running with the Console window in front, **then** you'll see the text "Ready..." in the Console window on the Macintosh. If sound is enabled, the device clicks.
6. To verify that the Palm OS device is responding to commands from the Macintosh, type into the Console window

   ```
   dir 0
   ```

   then press the Enter (not Return) key or Cmd-Return.

   A list of databases on the device appears in the Console window. If something is wrong, you'll notice a few second delay, then a message like `### Error $00000404 occurred`.

NOTE: If the Console window is not the active window when you enter the ".2" shortcut, you'll miss the "Ready..." message but will still be able to enter commands in the Console window.

Entering the ".2" shortcut sequence a second time does **not** print another "Ready..." message. If you're not sure whether the connection is established, enter `dir 0` in the Console window and see if it works.

Loading Applications Onto the Device

Before you can load an application onto the Palm OS device, you must first build an executable for the device. While building an application for the Simulator involves using a CodeWarrior project, building one for the Palm OS device itself requires using a Makefile (see Building a Palm OS Application).

The sample Makefile automatically copies the application into the Debugger folder. If your Makefile doesn't do this, you have to copy your application into the Debugger folder before proceeding.

1. If the Palm OS device is already running an old copy of your application, type `del 0 appname` into the Debugger Console window to remove it.
2. In the Console window of the Pilot Debugger application, type

   ```
   import 0 appname
   ```

   and press the Enter key.

   appname is the **case-sensitive** name of your application. If the name contains spaces, enclose it in double quotes.

   In the Console window, output like the following should appear:

   ```
   Creating Database on card 0
   name: HwrTest
   type appl, creator hwrt
   Importing resource 'code'=1....
   Importing resource 'code'=0....
   Importing resource 'data'=0....
   Importing resource 'MBAR'=1000....
   Importing resource 'tFRM'=1000....
   Importing resource 'tFRM'=1900....
   Importing resource 'tAIB'=1000....
   Importing resource 'tver'=1....
   Importing resource 'tAIN'=1000....
   Importing resource 'pref'=0....
   Success!!
   ```

   Your application is now loaded onto the Palm OS device and you can switch to it using the launcher button on the device.

Copying a Database From the Device to the Macintosh

To copy either an application database or a data database from the Palm OS device to the Macintosh, follow these steps:

1. In the Console window of the Pilot Debugger, enter the `dir 0` command to get a list of databases on the device and verify the name of the database you wish to copy.
2. In the Console window of the Pilot Debugger, type `export 0 <dbname>` where `<dbname>` is the case-sensitive name of the database you wish to copy and press the Enter key. (Don't confuse the application with the database.) If `<dbname>` has one or more spaces in it, enclose it in double quotes.

   In the Console window, you should see output like the following:

   ```
   Exporting record #0....
   Exporting record #1....
   Success!!
   ```

   The database has been copied into the Debugger folder under the same name as it had on the device. If you wish, you can now import this database into your simulated application on the Macintosh as described in Importing a Database Into the Simulator.

Importing a Database Into the Simulator

While debugging your application, you may at times find it helpful to take a database from the device and load it into the Pilot Simulator on the Macintosh. This lets you single-step through the source code of your application while it is working with data that originated on the device.

First, you must get the database off of the device by following the instructions in Copying a Database From the Device to the Macintosh. Then follow these steps to load the database into the Simulator:

1. Copy the database file into your application's folder on the Macintosh. The application must have been compiled for use with the Simulator. If you've just exported it from the device,

you'll have to copy it out of the Debugger folder and into your simulated application's root folder.

2. Launch your simulated application.
3. From the Window menu, select Console.

   This opens a Simulator Console window.
4. In the Console window, type
   ```
   opened
   ```
   then press the Enter key.

   This prints a list of open databases in the Simulator. If the database you want to import is already in the Simulator's memory image and is open, you have to close it first. If you see that it is open, enter
   ```
   close <accessP>
   ```
   into the Console window, where <accessP> is the access pointer of the database that is displayed by the `opened` command. For example, you might enter: `close 0054280C` if 0054280C was the access pointer of the database.
5. In the Console window, type
   ```
   import 0 <dbname>
   ```
   where `<dbname>` is the name of the database you want to import.
6. If you had to close the old database before importing the new one (described in step 4) **save the memory card image and quit the Simulator**.

   This closes the database belonging to your application, and it's not possible to force it to use the new database without restarting the application.

Closing the Simulator and restarting it allows the application to start fresh with the replacement database. To save the memory card image, follow these steps:

1. From the File menu, choose Save Card 0.
2. When prompted for the location, choose the current folder and the default name (Pilot Card 0), then click the Save button.
3. After saving, quit the Simulator and relaunch it to use the new database.

When you have a saved memory card image in the same folder as your application, the Simulator uses it the next time it launches.

Consequently, all databases that exist in the saved memory card image reappear the next time you launch your application.

Exporting a Database From the Simulator

Occasionally, it is helpful to take a database generated in your simulated application and write it out as a separate Macintosh file. You can then take this file and import it into a real device using the Pilot Debugger, or look at it using a Macintosh file utility.

1. Launch your application (which has to be compiled for use with the Simulator).
2. From the Window menu, select Console.
   This opens a Simulator Console window.
3. In the Console window, enter the command
   ```
   dir 0
   ```
   to get a list of databases in the Simulator's memory card image.
4. Verify the name of the database that you wish to export.
5. In the Console window, enter the command
   ```
   export 0 <dbname>
   ```
   where <dbname> is the case-sensitive name of the database you wish to export, and press the Enter key. (Use double quotes if the name contains spaces.)
   In the Console window, output like the following should appear:
   ```
   Exporting record #0....
   Exporting record #1....
   Success!!
   ```
   The database is now copied into the application folder under the same name it had in the Simulator's memory card image. If you wish, you can now import this database into your Palm OS device using the `import` command from the Pilot Debugger's Console window.

4

Design, Testing, and Localization Guidelines

This chapter helps you develop an application that's fast, robust, and consistent with other applications on the device.

The information was collected from engineers, testers, and other experts who designed, developed, and tested the four applications shipped with the first Palm OS device. A section on localization helps those developers that intend to make their application available in other languages.

The chapter discusses these topics:

- Application Design Guidelines
- Localizing a Palm OS Application
- Testing a Palm OS Application

Application Design Guidelines

This section provides some application design guidelines. It discusses these topics:

- Getting Started
- User Interface Do's and Don'ts
- Program Design
- Working With Databases

Getting Started

As you're starting to design your Palm OS application, keep in mind the differences between a hand-held device and a desktop computer. Note that these differences are not just restrictions, but reflect what your users want to do with the device. For a successful, easy to use application, follow these guidelines:

- Remember that the Palm OS device is meant for data viewing and gathering, not for data processing.
- Off-load all heavy-duty processing onto the desktop computer.
- Design for the small square screen. Strip all complexity from the user interface that you possibly can. Examine each feature on the desktop and ask yourself whether it's really needed, then try to implement it if it is.
- If you do decide to implement a feature, remember that easy access by one tap can reduce the complexity of your user interface.

User Interface Do's and Don'ts

After you've decided how your application will appear on the Palm OS device, the next step is creating a preliminary user interface. All Palm OS applications are based on a set of resource templates included with your development environment. You design your application's interface by filling in the templates appropriately.

A detailed specification of all user interface guidelines is provided in User Interface Design Guidelines.

NOTE: All developers are urged to include the first set of rules listed here in their test plan. Applications that don't follow them may cause problems for other applications on the device.

Here's a summary of the most important rules you should follow to make your application consistent with other Palm OS applications:

- Whenever a field for user input is available, make sure that:
  - system keyboard is available via shortcut
  - system keyboard is available via menu
  - Graffiti input is possible (regular strokes and shortcuts)
  - cut, copy, paste, and undo are possible
- Don't gray-out unusable menu items.
- Don't change or obscure the Graffiti status indicator area.
- Don't change or obscure the behavior of the silk-screened icons.

- Some applications (such as games) may find it useful to overload the buttons. If you do that, release the buttons at every possible opportunity.
- Be sure to handle the clipboard correctly. If you use it, allow users to copy and paste between applications; if you don't, make sure it's intact when you exit.
- Don't nest dialog boxes too deeply.

Here are some additional recommendations that make your application easier to use:

- If you can, provide command strokes for each menu command.
- If you can, allow finger navigation. For finger navigation, buttons need to be big enough for the system to recognize which button has been pushed. This is done by the Palm OS system software.
- Provide help dialogs (tips) where possible.
- Support Graffiti navigation: left-right-forward-backward movement is part of a field's behavior. Your application should provide inter-field navigation if appropriate by allowing users to get to the next and previous screen using down/up and up/down keystrokes.

Program Design

After you have some rudimentary user interface components, you can start writing the program itself. As you do, remember to look at the tutorial examples and the source code for the resident applications for examples. In addition, follow the guidelines below.

Integrating With the Palm OS Environment

When users work with a Palm OS application, they expect to be able to switch to other applications, receive alarms, and so on. Your application will integrate well with others if you follow these guidelines:

- All applications should handle sysAppLaunchCmdNormalLaunch and handle or ignore other application launch codes as appropriate.

- Handle system preferences properly. System preferences determine the display of
  - date formats
  - time formats
  - number formats
  - first day of week (Sunday or Monday)

  Note that system preferences are independent of localization issues.
- Allow the system to post alarms, low-battery warnings, and system messages during synchronization.
- Store state information in the application preferences database not in the application record database. Call `PrfGetAppPreferences` and `PrfSetAppPreferences` to save and restore preferences. This is important if your application returns to the last displayed view by default.
- If your application uses the serial port, be sure to free it up whenever you no longer need it so that the HotSync application can use it.
- If your application supports private records, be sure they are unavailable to the global find when they should be hidden.
- The application name is defined in two places:
  - The application icon name (optional), is a string resource in the application's resource file (tAIN). It is used by the launcher screen, and is often shorter than the long name. Using the icon name is useful if you plan to localize your application.
  - The long name is specified in the Makefile (required) and used by HotSync, the About box, the Memory display, and the database header.

Avoiding Potential Pitfalls and Achieving Optimum Performance

Because the Palm OS device has limited heap space and storage, optimization is critical. The Palm OS currently has no wait cursor.

To make your application as fast and efficient as possible, optimize for heap space first, speed second, code size third, and follow these guidelines:

- Ask Palm Developer Support for a unique creator ID for your application and use that ID to avoid overwriting other applications' databases (email: devsupp@palm.com).
- Use only one event loop to simplify debugging.
- Allocate handles for your memory to avoid heap fragmentation.
- Sort on demand; don't keep different sort lists around. This makes your program simpler and requires less storage.
- Dynamic memory is a potential bottleneck. Don't put large structures on the stack.
- To conserve the battery, avoid continual polling. If your application is in a wait loop, poll at short intervals (for example, every tenth of a second) instead. The event loop of the Hardball example application included with your Palm OS SDK illustrates how to do this.
- Arrange subroutines within the application to avoid 32K jumps (detectable only when building for the device; doesn't show in Simulator).
- Version your applications and databases using `dmSetDatabasInfo` when creating a database. Databases default to version 0 if the version isn't explicitly set.
- Call `dmDatabaseInfo` to check the database version at application start-up.
- To have your application run well within the constraints of the limited dynamic heap (32K) follow these guidelines:
  - Allocate memory chunks instead of using global variables where possible.
  - Switch from one UI form to another instead of stacking up dialog boxes.
  - Edit database records in place; don't make extra copies on the dynamic heap.
- Your application only has 2K of stack space. Avoid placing large amounts of data on the stack; global variables are preferable to local variables (however, chunks are preferable to global variables).

Working With Databases

- Keep data in database records compact. To avoid performance problems, Palm OS databases are not compressed, but all data are tightly packed. This pays off for storage and during HotSync.
- All records in a database should be of the same type and format. This is not a requirement, but is highly recommended to avoid processing overhead.
- Be sure your application modifies the flags in the database header appropriately when the user deletes or otherwise modifies information.
- When the user deletes a record, be sure to remove all data from it but don't remove the record itself. That way, the Desktop application can retrieve the information that the record is deleted the next time there is a HotSync.
- Don't display deleted records.

Localizing a Palm OS Application

If you're planning to localize the software you're developing for the Palm OS device, start by looking at the localized versions of the four PIM applications on the device. Then plan your application's interface, keeping in mind localization issues.

General Localization Guidelines

When you start planning for the localized version of your application, follow these guidelines:

- When designing the layout of a screen, try to allow extra space in strings. Don't use the English language version of the software as a guide. If you can, use a larger dialog than the English version requires. Note that because of the particularly scarce screen real estate on the Palm OS device, you may have to use abbreviations at times.
- Don't put strings in code. Separate any language-dependent strings from the code itself. If you have to print directly to the screen, remember that a one-line warning or message in one language may need more than one line in another language.

- Using a fine granularity is usually helpful. You can then concatenate strings as needed (and in the order needed) to arrive at a correct translation.
- Consider using string templates. For example, the MemoPad application uses the template: Memo # of %. The application can replace # and % to change the text.

Localization for Palm OS

There are several issues you need to pay special attention to when localizing a Palm OS application.

- **Preferences**. Your application needs to check and use the settings users have chosen in the system preferences. The default preferences at startup are different for the different localized versions, though they can be overridden. The following settings are available:
  - Date formats
  - Time formats
  - Start day of week
  - Number formats

  Note that there is no currency setting. There are also no separate setting for long and short dates; the long date format is mapped to the short date format.
- **Translating Palm OS Resources**. Applications that use only standard Macintosh ResEdit resources can use certain standard translation tools such as Resourcer or AppleGlot to facilitate translation. Because Palm OS resources are proprietary, you can’t use these tools but have to translate each resource individually instead.
- **Font Issues**. A Palm OS application is developed on the Macintosh but uses Windows fonts. The conversion occurs automatically; don’t try to second-guess the Windows font mapping.
- Capital letters with accent marks may be the same size as the letters without (in effect, the letter itself becomes one pixel shorter).
- Remember that most resources, for example lists, fields, and tips scroll if you need more space.

Testing a Palm OS Application

Testing a Palm OS Application consists of two stages, discussed in the next two sections:

- Testing in Standalone Mode makes sure that your application runs error-free and is robust in standalone mode.
- Testing Application Integration makes sure that your application is integrated with the Palm OS environment.

Testing in Standalone Mode

Testing your application in standalone mode involves a few simple steps:

- Make sure your application runs error free when you use the Pilot Simulator (see Using the Pilot Simulator).
- When you can detect no more errors, run the Gremlins automated testing application available from the Simulator. Your application should be able to pass at least 500,000 Gremlin events. If your application can handle one million events, it is considered robust.
- Run Gremlins with `mdebug` set to full. To do that, type `mdebug -full` into the console window, then run one of the Gremlins.
- Look for memory leakage after running Gremlins with the command `hd 0` (see Manipulating Memory in Using the Pilot Simulator).
- Ensure correct behavior in out of memory conditions by using heap fill, heap compact, heap scramble, and heap check. See Debugging Memory Management.
- Perform limits testing:
  - Fill up memory and see if the application still runs.
  - Make sure you handle a storage-full situation properly. The application should be able to detect that there's no memory left, and should warn the user and not attempt to write records to a database.
- Finally, download your application to the Palm OS device and perform testing there. Start by continuing testing in standalone mode, then test its integration with the Palm OS environment and with other applications.

Testing Application Integration

A Palm OS application needs to fit in with the environment in a way that users expect. For example, users need instant access to other applications such as the calculator; they expect to have access to Graffiti and the on-screen keyboard; and they expect to access information with the global find. Here are some guidelines:

- Be sure your application does not obscure or change the Graffiti area, silk-screened buttons, and power button.
- Follow the guidelines listed in User Interface Design Guidelines and pay special attention to these points:
  - Test that the different user input modes (e.g. Graffiti and keyboard) are available for each field.
  - Test that menu items work with shortcuts as advertised.
  - Put limits on the length of fields and test them.
  - Test that any growable control, such as the launcher window or the popdown menus, scrolls correctly.
- Be sure your application uses the System Preferences for numeric formats, date, time, and start day of week. Do this even if you're not planning on localizing your software.
- Test that your application handles system messages during and after synchronization properly.
- Don't display deleted records.
- Test that your application doesn't exceed the maximum number of categories.
- Test that your application handles the global find properly. Generally searches and sorts aren't case sensitive.
- If your application allows for private records, test that they are hidden properly and that a global find ignores them.
- Always leave the Graffiti icon area free; don't obscure Graffiti shift indicators.
- Test that your application uses a consistent default state when the user enters it:
  - Some applications have a fixed default; for example, the Date Book always displays the current day when launched.
  - Other applications return to the place the user exited last. In that case, remember to have a default if that place is no

longer available. Because of HotSync and Preferences, don't assume the application data is the same it was when the user looked at it last.

- Test for performance. Launching, switching, and finding should be fast.
- If your application uses sounds, be sure it uses the Warning and Confirmation sounds properly.

5

User Interface Design Guidelines

The Palm OS device is designed for rapid entry and quick retrieval of information. To maximize performance, the UI should minimize navigation between windows, opening of dialog boxes, and so on. The layout of application screens needs to be simple so that the user can pick up the product and use it effectively after a short time. It's especially helpful if the UI of your application is consistent with other applications on the device so users work with familiar patterns.

This chapter helps you design a user interface that's intuitive, easy to use, and consistent with other applications on the device. It discusses these topics:

- Foundations of UI Design
- User Interface Objects Overview
- UI Design Philosophy

NOTE: Guidelines for implementing specific user-interface objects, such as information on the size of buttons or the font for labels, is provided in Chapter 3, "Palm OS User Interface Resources," of "Developing Palm OS Applications."

Foundations of UI Design

Users of the Palm OS device operate it with a plastic stylus or their fingers. Four buttons provide access to the Personal Information Management (PIM) applications (Date Book, To Do list, Address Book and Memo Pad). Here are some guidelines you should follow to make your application's interface consistent with these and other applications.

- Provide an application icon for the Launcher. When users want to open your application, they go to the launcher screen and select the icon.
- Your application probably needs a base screen that provides an overview of all available information. This screen is typically a list view.
- Allow users to view most record information by pressing the navigation keys. Each event, to do item, address, memo page, and so on, is called a record.
- For more efficient screen use, consider organizing records into user-defined categories. Users can switch between categories using a popup menu or can display all records at once.
- Detailed information and advanced navigation require the use of a stylus. See Data Entry for different data entry modes.
- Don't require double taps.
- Don't use grayed-out menu commands; instead, remove the command when it's not available.
- Use bold font for labels, nonbold for editable items.

Navigation

Users can move through an applications by the following methods:

- **Switching applications**. Users press the physical buttons representing the PIM applications, or access a launcher to switch applications.
- **Switching views**. Each PIM application has two or more views (or modes), typically a list view (or view mode) and an edit view (or edit mode). The user taps on records or uses command buttons to toggle between these views.

  Edit mode gives users access to the Details button, for settings which affect the entire record. They can also access specific menu commands for records. In many applications, tapping on a record switches the application to edit mode and displays an input cursor.
- **Switching categories of records**. A popup menu lets users switch between categories. The popup menu is found in the list view of applications that support categories.

- **Switching records in applications**. Depending on the application, the user can scroll through lists of records, and tap on the records or a Details button for further information.
- **Scrolling**. Records too long to display in one screen are scrollable. On-screen scroll buttons allow users to move up or down one line at a time. The physical arrow buttons allow users to move up and down one page at a time.

Data Entry

Users can enter data, which is stored at the application and system level, as follows:

- **Graffiti.** Graffiti characters are written in the text area on the digitizer and appear on the screen at the cursor location. The user places the cursor directly on the screen with the stylus.
- **On-screen keyboard**. In place of Graffiti, the user taps with the stylus an on-screen keyboard that types text into a temporary window. When the user dismisses the keyboard, the system inserts that text at the cursor location.
- **Controls**. Buttons, check boxes, and popup lists provide a quick way to enter settings and select options.
- **HotSync**. The user can type data on the PC and download it to the Palm OS device.

In addition, follow these guidelines when designing the data entry interface for your application:

- Let users perform basic data entry in place.
- Provide a Details dialog for more elaborate data entry.
- In the Details dialog, use this format:
  ```
  Item (right-justified):Value(left-justified)
  ```
  for example:
  ```
  Date:4-1-96
  ```
- Allow users to interact with an application through either a button, menu, or popup list. Don’t provide both a button and a menu for the same actions.

Command Execution

Users can execute commands as follows:

- **Command buttons**. Users execute common commands by tapping on command buttons at the bottom of the screen.
- **Menus**. Commands not represented by command buttons can be accessed via a simple menu system. The user taps on a menu hard icon in the digitizer area to invoke a menu bar. Provide menu shortcuts if possible.
- **Graffiti menu command shortcuts**. Users can write a special Graffiti stroke and a command keystroke to execute a menu command. This is analogous to keyboard shortcuts on a personal computer. For example, writing the command stroke symbol (a bottom-left to top-right line) and “C” allows the user to copy the selected text.

Screen Layout

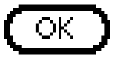
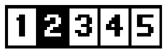
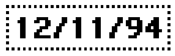
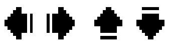
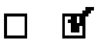
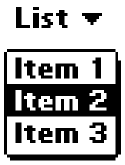
When designing the screen layout for your application, follow these guidelines:

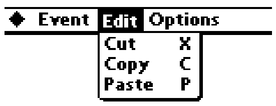
- Use the resources provided with the development environment and use the values for width, height, default font, and so on, provided in Chapter 3, “User Interface Resources” of “Developing Palm OS Applications.”
- Always go to the edge of the screen (don’t use borders). This maximizes screen real estate available to the application.
- In the title bar for each screen, provide both the application name and the name of the screen (if possible).
- When your application brings up a dialog, align it with the bottom of the screen. Leave the screen Title bar visible if possible.
- Align buttons at the bottom of the screen with the bottom edge of the screen.
- When using buttons in dialogs, leave space (three pixels) between the edge of the dialog and the buttons.
- Text surrounded by borders normally has one pixel above and below the font height.
- Controls that can be displayed in groups have at least two pixels to the left and right of the text label (the exception is

command buttons, which require wider margins to accommodate the rounded border).

User Interface Objects Overview

The Palm OS development environment provides a set of Macintosh resource templates that application developers use to implement the buttons, dialogs, and other UI elements. The table below maps user interface element to resources. All resources are discussed in detail in Chapter 3, "Palm OS User Interface Resources," of "Developing Palm OS Applications, Part I." Specific design recommendations for some of the elements are provided below.

UI Element and Functionality	**Example**	**Resource**
Command button Execute command.		Button (tBTN)
Push buttons Select a value (same as radio buttons).		Push button (tPBN)
Hot text entry Invoke dialog that changes text of the button.		Selector trigger (tSLT)
Increment arrows Increment/decrement values, or scroll.		Button (tBTN) or repeating button (tREP)
Check box Toggle on or off.		Checkbox (tCBX)
Popup lists Choose a setting from a list.		Popup trigger (tPUT) Popup list (tPUL) List (tLST)

UI Element and Functionality	Example	Resource
Menus Execute commands not found on screen.		Menu Bar (MBAR) Menu (MENU)
Text fields Display text (single or multiple lines).		Field resource (tFLD)

UI Design Philosophy

This section looks at some issues that underlie the design of a user interface for the Palm OS Device. It discusses these topics:

- Creating Fast Applications
- Matching Use Frequency and Accessibility
- Creating Easy to Use Applications

Creating Fast Applications

Speed is a critical design objective for hand-held organizers, and is not limited to execution speed of the code. The total time needed to navigate, select, and execute commands can have tremendous impact on overall efficiency.

The user should be able to keep up with someone on the telephone when setting up appointments, looking up phone numbers, etc. Priorities include the ability to

- execute key commands quickly
- navigate to key screens quickly
- find key data quickly (for example phone numbers)

Matching Use Frequency and Accessibility

PC user interfaces are typically designed to display commands as if they were used equally. In reality, usage of commands tends to drop off rapidly after a few key commands. Similarly, some settings are more likely to be used than others (for example, a 3 PM to 4 PM meeting occurs much more frequently than a 3:25 to 4:15 meeting). More frequently used commands and settings should be easier to find and faster to execute.

- Software commands executed with extreme frequency should be accessible by one tap.
- Infrequently used commands may require more user actions to execute.

Frequency	**Example**	**Accessibility**
Several times per hour.	Checking today's schedule or to do items.	One tap.
Several times per day.	One hour meeting starting at the top of the hour.	One tap, write in place.
Several times per week.	Setting a weekly meeting (repeating event).	Several taps, second dialog box.

To make something more easily accessible, follow these guidelines:

- Minimize the number of taps to execute a function or make a setting.
- Commonly executed multistep operations, such as going to today, should have command buttons to streamline execution.
- Minimize the need to change screens.
- Minimize the number of dialogs to open and close; avoid dialogs within dialogs.

Choose the appropriate UI object when making a speed vs. screen layout decision:

- Buttons on the screen provide instant access but take up valuable screen space.
- Push buttons are faster than popup lists and should be used if they fit on the screen reasonably.
- Popup lists are faster than manual input or increment/decrement buttons but can be cumbersome if there are too many items on the list or if they need to scroll.

Creating Easy to Use Applications

The user must be able to pick the product up and, with no training or instruction, navigate between applications (without getting stuck) and execute basic commands within five minutes. Advanced commands should be easily accessible but should not be in the way. The design must therefore fit the following criteria:

- Users need to know where they are. The PIM applications and modal dialog boxes have black title bars that indicate the application name and view.
- Make it obvious to the user how to get to different views. The command buttons provide the best example of achieving this.
- Use buttons for important commands.
- Accomplishing common tasks should be fast and easy. Minimizing steps helps not only speed but ease of use.

Ease of use amounts to a series of trade-offs. Striking the best balance for the most people is the biggest challenge of UI design. For example:

- The fewer buttons on the screen, the less time it takes to learn how to use the product. However, having a button on screen means the user does not need to memorize how to find it. Keeping a few frequently used buttons on screen helps reduce the time spent learning basic functionality.
- Advanced features should not be in the way for beginners, but should not require multiple-step searching.
- Consistency reduces the time needed to learn an application by limiting the number of things that people need to keep in their head at once. If the up arrow key does different things on different screens, for example, the user needs to memorize an entire set of rules to use the device easily.