Welcome to the

Palm OS Tutorial

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 Table of Contents entry or use the arrows in the menu bar**

U.S. Robotics®

Palm OS™ Tutorial

Some information in this manual may be out of date.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

Introduction

Overview

The goal of this tutorial is to lead you, the developer, through the process of creating and testing a sample Palm OS application. The sample app is a memo pad, similar to a Palm OS's built-in Memo Pad. In doing so, you will learn the basic techniques of Palm OS programming, using the Pilot Simulator, and debugging.

The tutorial is presented as a series of phases, each phase contains a series of steps. Each phase creates a functioning application with capabilities that build on and extend those of the previous phase. Each step describes what to do and why. Source code for each completed phase is included, so that you can build each phase independently. You'll find the tutorial illustrated throughout with the source code and resources of the sample application, and diagrams where appropriate.

To work the tutorial, you must have installed the Macintosh-based development environment (see Release Notes and installation instructions). To develop software in this environment, you should be an experienced C or C++ programmer, familiar with the Macintosh and its development tools. A review of "Application Control Flow Overview" in the **Developing Palm OS Applications Part 1** document is highly recommended.

0.1 System Requirements

To run the Palm OS SDK and tutorial you need the following:

- Hardware
- Software

0.1.1 Hardware Requirements

- CPU: Macintosh Quadra 700 or "better." PowerPCs are fine.

- RAM: 16MB recommended
- Disk space: 40+MB

0.1.2 Software Requirements

- Apple System 7.1 or later with a 68030 or 68040 processor.
- Apple System 7.1.2 or later for the PowerPC 601 and 604 processors.
- Metrowerks CodeWarrior version 9.0 or greater
- Apple ResEdit 2.1.x (comes with CodeWarrior)
- Apple Macintosh Programmer's Workshop (MPW - also supplied with CodeWarriorMPW)

0.2 Installation: The Folder Structure

This section describes the Macintosh folder structure and installation procedures for loading the Palm OS SDK.

To work through the Palm OS SDK tutorial and build Palm OS applications, the following folder structure has been created:

Palm OS SDK

Apps

Debugger

Docs

Examples

Incs

Libraries

Tools

Tutorial

The Tutorial folder contains all the source files for each phase. These source code files serve as a backup and standard against which you can check your work. The "Working Folder", located in the Apps folder is where you will build each phase of this tutorial. Your work for each phase will overwrite your work for the previous phase.

0.3 Navigating Tutorial Phases

It is recommended that you work through all of the phases in order, to get the best sense of how an application is built. However, once you've finished the tutorial, you may want random access to various phases. The folder structure is set up to allow this. Generally, to set up any particular phase of the tutorial after 0, follow these steps:

1. Copy the MemoPad.c, MemoPadRsc.c, and MemoPadRsc.h files from the desired Tutorial:MemoPad <n>:Src folder into Working Folder:Src
2. Copy the MemoPadMain.rsrc resource file from the Tutorial MemoPad <n-1>:Rsc folder into Working Folder:Rsc
3. Copy the Tutorial:MemoPad <n-1>:MemoPad.μ project file into Working Folder.

Creating Memo Pad Form and Button

Overview

Phase 1 differs from other phases in that you must create a CodeWarrior project. The purpose is to give you the experience of creating all the pieces of a project except the source code. If you've never worked in the Macintosh environment or with ResEdit, going thorugh this process is a big help. When you begin creating your own projects, you can use Phase 1 as a model for fitting the pieces together. The basic steps in Phase 1 of this tutorial are:

- Creating a Desktop Project Folder
- Building a ResEdit Resource File
- Modifying a CodeWarrior Project
- Setting File Access Paths
- Examining the Code
- Building and Exercising the Project Application

Convention: Although Phase 1 reminds you to run CodeWarrior or ResEdit before proceeding with a series of steps, the rest of the phases of the tutorial assume that you have these tools running.

1.1 Creating a Desktop Project Folder

To contain all the pieces of programming project, we set up a series of Macintosh desktop folders—one for each phase of the project. Take a look:

1. From the Finder, open the Palm OS SDK folder.
2. Navigate to the start of the tutorial. Follow this path on your hard disk—Palm OS SDK:Tutorial:MemoPad 01:Src.
3. Copy these three source files into the Src folder of your working folder, located at Palm OS SDK:Apps:Working Folder:

   - MemoPad.c
   - MemoPadRsc.c
   - MemoPadRsc.h
4. Copy the project file from Palm OS SDK:Tutorial:MemoPad 00:MemoPad.μ to the Working Folder.

Tip: To make life easier for yourself, put aliases to ResEdit and Code Warrior.IDE 1.4 in the folder where you're building your project, or on the Apple menu, or on the desktop. (If you're unfamiliar with aliases, see your "Macintosh Reference" for details on the File>Make Alias command.)

1.2 Building a ResEdit Resource File

The first task is to build your resources. (A resource is a statically initialized UI object or string.) Begin by creating a resource file for the MemoPad project; follow these steps:

1. Launch ResEdit.
2. Create a ResEdit file to hold project resources (File>New).
3. In the file dialog, select a file path to the desired project folder, Palm OS SDK:Apps:Working Folder:Rsc.
4. Enter the resource file name MemoPadMain.rsrc and click New.

MemoPadMain.rsrc is an empty container that you will fill with the resources you create for the first screen of the MemoPad application.

1.2.1 Creating the MemoPad Form, Title, and Button Resources

Figure 1.1 shows the memo pad you're creating. Study its components:

- A main window or form, the basic object that contains the other components
- A title (MemoPad)
- An Exit button

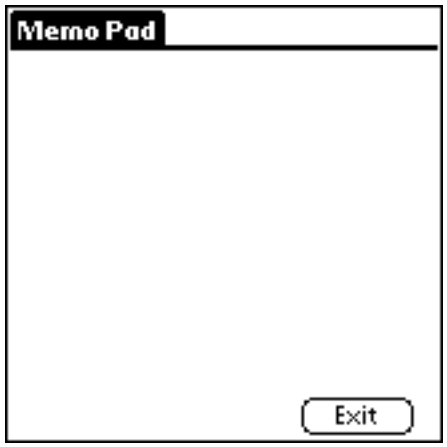

Figure 1.1 The Phase 1 Memo Pad

You can create each of these resources (and many others) quickly and easily with ResEdit. This section shows you how to create the resources needed for MemoPad 1, beginning with the form. The form is the visible area that acts as a container for other user interface objects in the project—the title and the button.

1.2.1.1 Creating the Main Form Resource

With ResEdit running and the MemoPadMain.rsrc resource file open, create a form (tFRM) resource for the memo pad. Choose Resource>Create New Resource (CMD-K). Create a form resource; in the Select New Type dialog, enter tFRM, and click OK. See "Resource Cookbook" for details.

Fill in the edit form for the resource as follows:

1. Set the dimensions to 0, 0, 160, 160.
2. Set the radio buttons as follows:
   - Usable radio button (True)
   - Modal radio button (False)
   - Save behind radio button (False)
3. Set Form ID field to an ID number of 1000. See the section, "Notes on Resource Numbering," below for details. You must change this ID in two places—in the this field and in Resource Info (see the next section).

4. To complete the form attributes, type a 0 into the Help Rsc ID, Menu Rsc ID, and Default Button ID boxes.
5. Skip the Number of Objs attribute. ResEdit automatically fills that in for you as you add objects (like the title and button) to the form.

1.2.1.2 Setting a Name for the Form

Palm OS uses the ID number you set in the editor window. You need to set a name for the resource and an ID number for the Macintosh system. Later, in the sections Examining MemoPadRsc.c and Examining MemoPadRsc.h, you'll see how code uses these numbers to manipulate resources. Set a name and ID number for the main form; follow these steps:

1. Open the Resource Info dialog (CMD-I).
2. Assign an ID number; type 1000 into the Form ID box (replacing 128).
   For debugging purposes, keep this number the same as the ID number in the tFRM edit window.
3. Set the name to "Memo Pad Main".
4. Close the Info window.
5. Close the editor window.
   The window in the forefront lists the one resource you have created. ResEdit users call this a "resource picker" or "resource picker window." You'll add more form resources to this project in later phases and then pick them from it.
6. Close the resource picker.

   The remaining window, representing the different types of resources in the project, is called the "type picker window" or "type picker." In the next section, you'll add two resources to it. Do not close the type picker.

1.2.1.3 Creating the MemoPad Title Resource

Resource creation, regardless of the type, follows a pattern. In this section, you create a title (a tTTL resource) for the memo pad. Follow these steps:

1. Within MemoPadMain.rsrc, create a title (tTTL) resource (CMD-K). See the "Resource Cookbook" for details.
2. In the editor window, type the title "Memo Pad".

3. Set resource Info (CMD-I). Enter 1001 for the ID and "Memo Pad" as the name.
4. Close the all windows back to the MemoPadMain.rsrc type picker.

1.2.1.4 Creating the MemoPad Button

Within MemoPadMain.rsrc, create a button (tBTN) resource (CMD-K). See the "Resource Cookbook" for details.

Create the button (a tBTN resource), follow these steps:

1. Continue the consecutive numbering of IDs. Give the Button ID as 1002.
2. Specify button position and size as 110, 146, 40, and 12.
3. Leave all radio button options set to their default: True
4. Specify the default font; type 0 (zero) for the font number.
5. Give the button a label; type "Exit".
6. Set resource Info (CMD-I). Enter 1002 for the ID and "Exit" as the name.
7. Close all the windows back to the MemoPadMain.rsrc type picker.

1.2.2 Notes on Resource Numbering

If you don't have any resource numbering conventions try ours:

- Form IDs, beginning with 1000, are multiples of 100.
- Only system resources get ID's less than 1000.
- Each user interface (UI) ID within a particular form begins with the number of its form.
- All objects in application/project have a unique resource ID number. **Note**: Resources of different types can have the same ID, we recommend against this practice. In complex projects, non-unique IDs can create confusion.

Here are a couple of numbering examples:

  - Form ID = 1000. Its button ID = 1001. Its title ID = 1002.
  - Form ID = 1100. Its button ID = 1101. Its title ID = 1102.

1.2.3 Attaching Resources to a Form

This section shows you how to put the title and button resources into an array in the form resource; follow these steps:

1. Go to the MemoPadMain.rsrc type picker and open the Form resource picker, the tFRM item.
2. From the resource picker, open the main form (ID 1000) editor window.
3. The tFRM resource lets you build an array of objects that the form itself is to contain. Look below the label "Number of Objs" (skipped earlier) and click the item labeled "1) ******".
4. Create a field for the title (CMD-K). Notice that ResEdit incremented the Number of Objs to one.
5. You may need to scroll the editor window a bit to do the next step. Put the title resource in the field. Set the ID to 1001 and the Type to tTTL.
6. Click the item labeled "2) ******"
7. Create a field for the button. Notice that the field Number of Objs is 2.
8. Scroll, if necessary, and fill in the information for the button resource. Set the ID to 1002 and the Type to tBTN. See Figure 1.2 below.
9. Close the all windows back to the MemoPadMain.rsrc type picker.
10. Save the resource file.

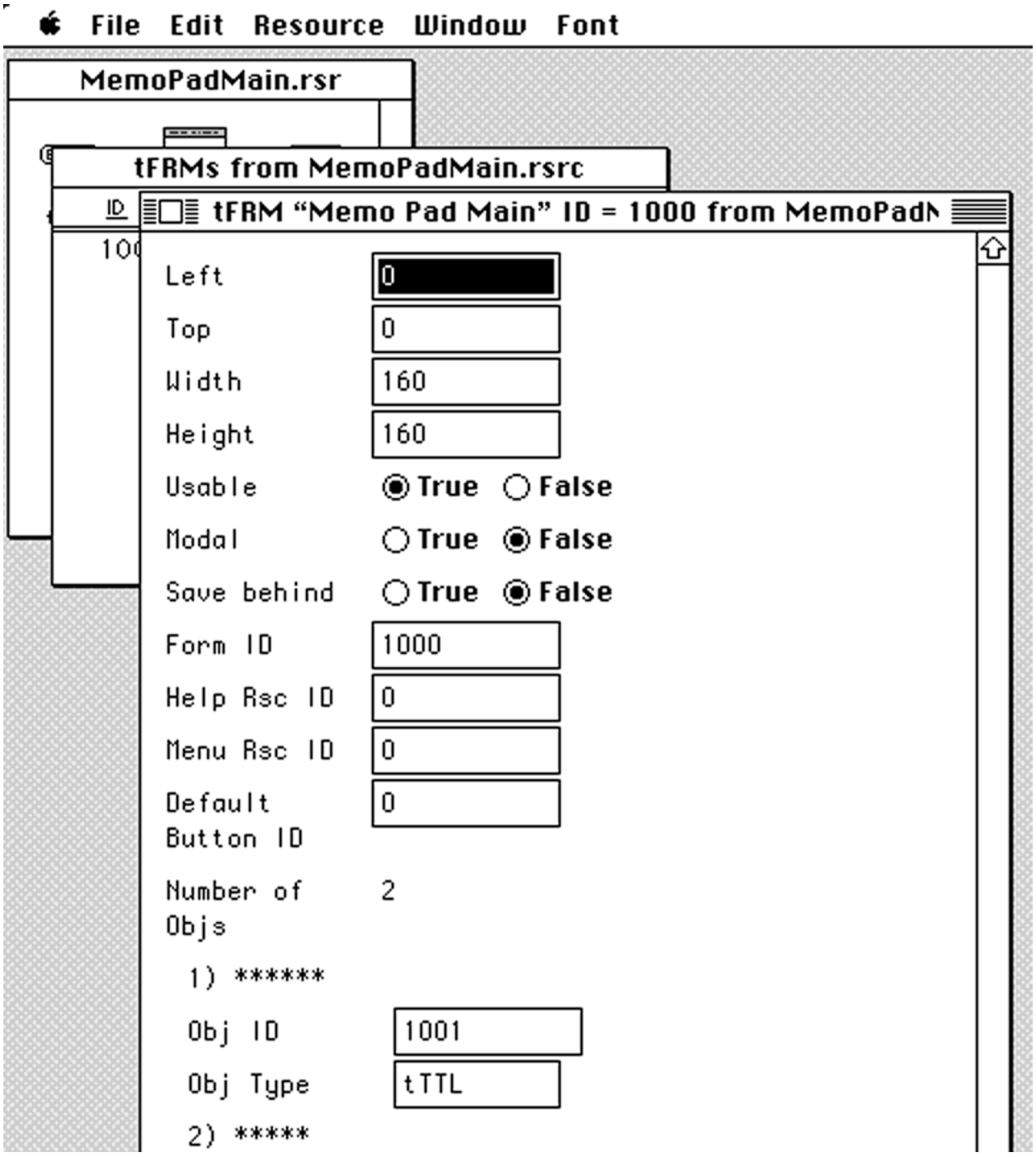

Figure 1.2 The tFRM editor window with resources filled in.

1.3 Modifying a CodeWarrior Project

The tutorial starts you with basic source files (MemoPad.c, MemoPad.h, MemoPadrsc.c). To that you've added a ResEdit resource file (MemoPadMain.rsrc). This section shows you how to modify a basic CodeWarrior project file. Double-click on MemoPad.μ in the Working Folder. This will load CodeWarrior with the project file MemoPad.μ opened.

1.3.1 Adding Files to the Project

The Project window has some placeholders: <replace me Mac>.c for code and <replace me>.rsrc under Resources. The typical pattern is to load the files you want for your project and then remove placeholders; follow these steps:

1. Select the Application file area, by selecting the placeholder, <replace me Mac>.c.
2. Choose Project>Add files.
3. Navigate to the Working Folder:Src folder.
4. From the dialog box, select the following source (.c) files to add, and click Add, or Double-click on the files:
   - MemoPad.c
   - MemoPadRsc.c
5. When you have clicked in the files from the list, click Done.
6. Remove the placeholder. Select <replace me Mac>.c and choose Project>Remove Files.

NOTE: You don't need to specify header (.h) files like MemoPadRsc.h in this process. CodeWarrior is smart enough to find these, based on your included specifications.

To load resources into the project, follow these steps:

1. Select the Resources file area by selecting the placeholder, <replace me>.rsrc.
2. Choose Project>Add files.
3. From the dialog box, select the resource file Working Folder:Rsc:MemoPadMain.rsrc

4. Click Add and then Done.
5. Remove the placeholder. Select <replace me>.rsrc and choose Project>Remove Files.

There are other files to be added to the project as the tutorial continues. Figure 1.3 shows a completed project window.

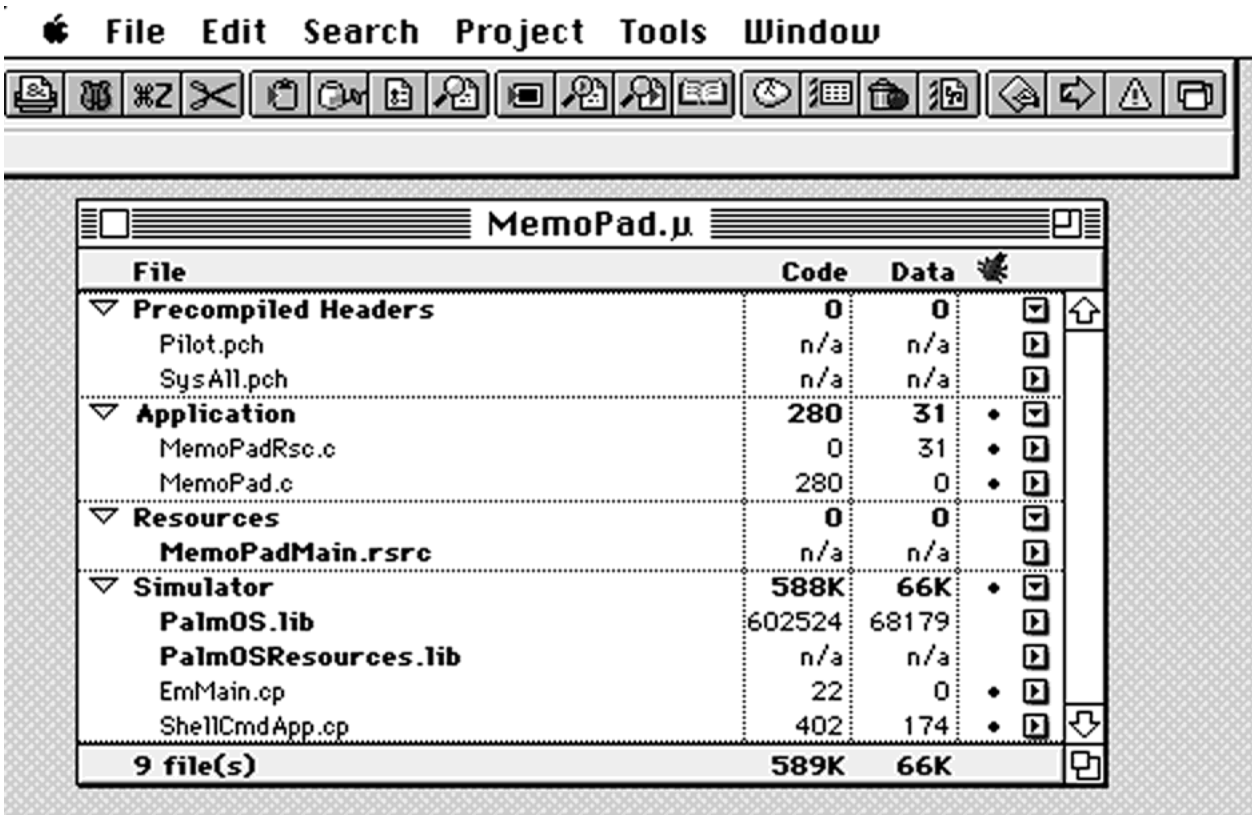

Figure 1.3 A completed CodeWarrior project window.

1.3.2 Assigning an Application Name

An application name has already been assigned to this project. To learn how it was done, follow these steps:

1. Choose Edit>Preferences.
2. Click the 68K Project icon.
3. Type the project name; MemoPad.mac.

4. For Creator, enter Pilot.
5. For Preferred Heap Size, enter 2048. For Minimum Heap Size, enter 2048. Your screen should resemble Figure 1.4.
6. Save your application references and return to the project; click OK.

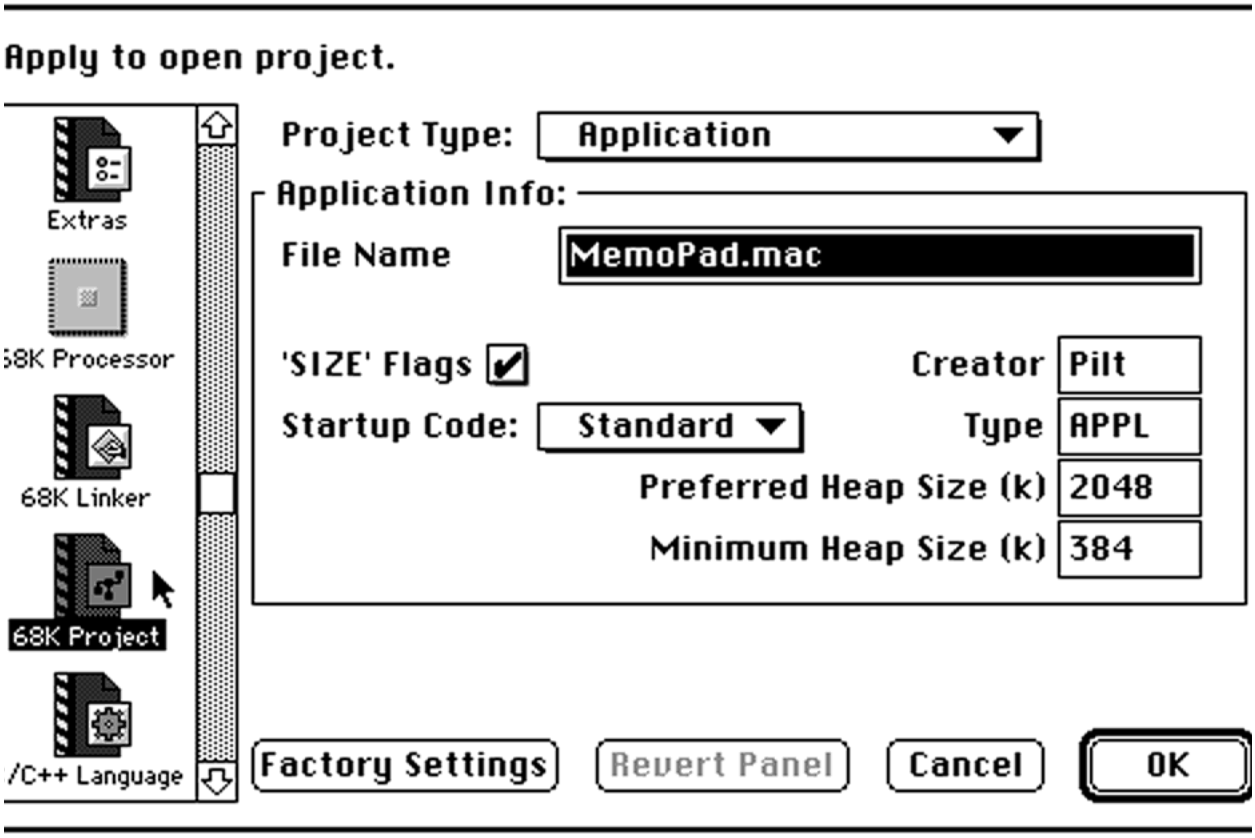

Figure 1.4 The 68K Project window completed.

1.4 Setting File Access Paths

If you move your project to a place other than the Apps:Working Folder folder, then you may have to reset the access paths. This can be done via the Edit>Preferences>Access Paths dialog. See the CodeWarrior documentation for further instructions.

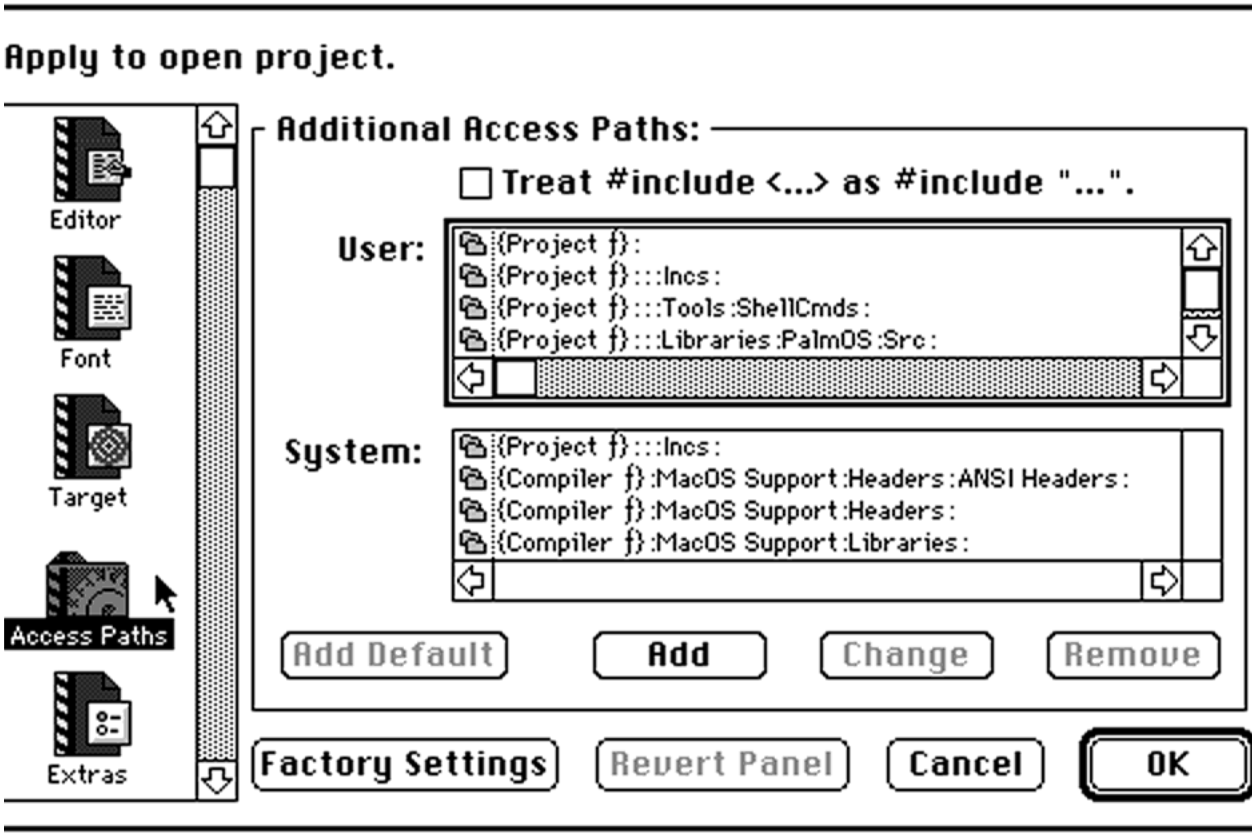

Figure 1.5 The Access Path window.

1.5 Examining the Code

Although the code for Phase 1 is fairly brief, it presents an accurate picture of the basic elements. The three key application code files are:

- MemoPad.c contains the C code for the application, including the event loop and key functions.
- MemoPadRsc.c identifies to the Pilot Simulator application which application resource files are required by the application at run time.

- MemoPadRsc.h contains #define macros for resource IDs that are used by the application. These macros map resource names to their IDs.

1.5.1 Examining MemoPad.c

From CodeWarrior, open MemoPad.c in the MemoPad 01 folder. Examine the basic application code structure. In the preprocessor area, the inclusion of the Pilot.h file includes all the header files for the Palm OS system functions and data structures.

Function prototypes for the file follow the macro definitions:

- StartApplication
- MainFormHandleEvent
- EventLoop

The PilotMain function drives these functions.

1.5.1.1 The PilotMain Function

The PilotMain function is at the end of the file. It serves as the entry point for the application. It takes the place of the "main()" entry point in traditional C programming. In many programs, you pass parameters to PilotMain that can influence its operation. In this simple case, these parameters are ignored. Take a look at PilotMain. It makes two calls—one to StartApplication and one to EventLoop.

1.5.1.2 The StartApplication Function

The StartApplication function initializes the required data structures or system facilities and initiates the application's user interface before the event loop begins processing events. In this simple example, the main form is loaded into memory and set as the active form. It is then drawn on the display.

1.5.1.3 The EventLoop Function

The EventLoop function (found just above PilotMain) retrieves the next available event from the system and passes it to a succession of event handlers for processing. Each handler can choose (by returning a non-zero value) to handle an event, so that it is not seen or processed by subsequent event handlers. The EventLoop function processes all the events that flow through the application as it exe-

cutes. EventLoop returns when an event is encountered that directs the application to quit. When control returns to PilotMain from EventLoop, the application quits by returning an error code of zero.

1.5.1.4 The SysHandleEvent Function

EventLoop gives the SysHandleEvent function the first opportunity to process each event. This function processes low-level events, like low battery conditions, application launch, power keys, and Graffiti recognition.

1.5.1.5 The MainFormHandleEvent Function

If SysHandleEvent doesn't handle the event (indicated by a return value of zero), it gets passed to the MainFormHandleEvent function. This function (found just above the EventLoop routine in the file) processes events related to the application's main form. The MainFormHandleEvent routine handles all events that require special processing for the main form. This simple example has only one event to be handled, ctlSelectEvent.

The presence of a ctlSelectEvent means that the user has tapped a button. The event structure contains the button's identity. However, there is currently only one button on the form, the Exit button. It causes the application to quit by creating a new appStopEvent and adding it to the system's event queue. The application retrieves this event later in the EventLoop routine. The MainFormHandleEvent routine returns a value of True, if no further processing is required for the event. This tells subsequent event handlers not to process the event.

1.5.1.6 The FrmHandleEvent Function

If MainFormHandleEvent doesn't handle the event, it gets passed to FrmHandleEvent function for default processing, if any, for every event. The event loop continues processing events, until it gets an appStopEvent. This event is the standard mechanism for causing the application to quit. When running an application from within the Pilot Simulator, the Pilot Simulator generates the appStopEvent, when you choose File>Quit.

1.5.2 Examining MemoPadRsc.c

With CodeWarrior still running, open MemoPadRsc.c and examine its file structure. This file defines the application and resources. The application's resources consist of the Main form and its components. These are contained in the MemoPadMain.rsrc file that you created earlier with ResEdit. This set of required resources grows as you add more resources to the application.

1.5.3 Examining MemoPadRsc.h

Open the MemoPadRsc.h file (located in the Src folder) and take a look at its contents. A shortcut for opening the .h file that corresponds to a .c file is to type CMD-TAB from the .c files window. The file includes definitions for all the resource ID's that constitute the application's interface to the outside world. The MemoPadRsc.h file lists two of the three resources:

```
#definemainForm1000
#definemainExitButton1002
```

The programmer didn't create a macro for the tTTL resource, because the title doesn't interact with the outside world. It belongs to the main form, which does the interacting.

These macro definitions make it possible to pass a resource ID to a function meaningfully, using the resource's name. Also notice the beginning of a convention here. The form name (main) comes first and then the resource name. Soon, you'll see new resources like editForm added to this list. Keep this convention in mind when you begin your own design and coding.

As a rule, for each resource that the application must manage, you must define the resource name and the corresponding resource ID. The ID must match the ID of the corresponding resource in the MemoPadMain.rsrc resource file. As more resources are added to the application, this list of macros grows.

1.6 Building and Exercising the Project Application

Now that you have a project, resources, and code for the first MemoPad project, you can build and run your project. Follow these steps:

1. With CodeWarrior running, check the Project menu to make sure that the debugger is disabled before running the application. It should show the item Enable Debugger. If the item says Disable Debugger, choose it.
2. Reset the file paths by choosing Project>Reset File Paths, and OK'ing the dialog.
3. Remove any binary files by choosing Project>Remove Binaries, and OK'ing the dialog.
4. Build the project (CMD-M).
5. Assuming that all goes well, run your MemoPad application (CMD-R). The Pilot Simulator starts and displays the Pilot with your memo pad application in it. It should resemble Figure 1.6.
6. To retire the MemoPad, click the Exit button you built. **Hint**: The Escape key also generates an appStopEvent.

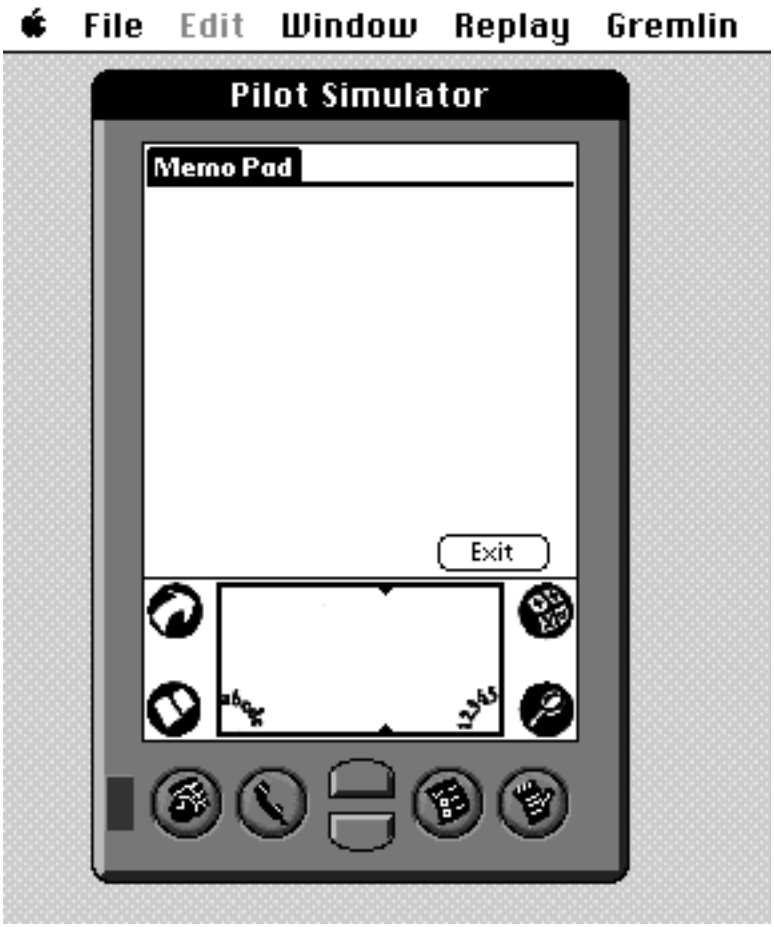

Figure 1.6 Phase 1 Code Running the Pilot Simulator.

Adding a Menu and a Simple Dialog

Overview

Phase 2 adds a menu with a Get Info item and an Info dialog that appears when you tap the menu item. See Figure 2.1 and Figure 2.2.

To set up the appropriate files for this phase, follow these steps:

1. If MemoPadMain.rsrc does not already exist in Working Folder:Rsc from Phase 1, then copy the MemoPadMain.rsrc resource file from the MemoPad 01:Rsc folder.
2. Copy MemoPad.c, MemoPadRsc.c, and MemoPadRsc.h from the MemoPad 02:Src folder to Working Folder:Src.
3. If MemoPad.µ does not exist in Working Folder from Phase 1, copy it from MemoPad 01.

Figure 2.1 Phase 2 Memo Pad with Get Info menu item open.

Figure 2.2 Phase 2 Memo Pad with the Get Info item displayed.

2.1 Adding a Main Menu with a Get Info Item

The key activity in Phase 2 is building a simple menu structure from which you can summon an Info dialog. This involves creating:

- Menu resources
- Get Info form resources
- Appropriate handlers and #defines to run the resources

2.1.1 Adding Resources for a Get Info Menu Item

In the world of UI menu construction, most systems follow a similar architecture:

- A form or window in which to anchor the menu system
- A menu bar to hold the various menus
- The menus themselves (File, Edit, Help, and so on)
- The menu items (New, Open, Close, Save, Quit, and so on)

Palm OS follows this same hierarchy, using ResEdit to construct a menu system. To build a menu within the main form (created in

Phase 1), first construct the individual components and then put them together. These components are:

- The Main Menu menu bar, an MBAR resource.
- The Options menu, a MENU resource.
- Within the Menu, you create menu items. This example has only one item.

2.1.1.1 Creating the Menu Bar

To begin, open the resource file for the main form in ResEdit:

1. Open the Working Folder:Rsc folder.
2. Open MemoPadMain.rsrc.

To create the menu bar (MBAR) resource, follow these steps see "Resource Cookbook" for details):

1. Create a new resource (CMD-K).
2. Enter MBAR as the resource type and click OK.
3. In the editor window, click the item labeled "1) ******"
4. Insert a new field (CMD-K).
5. Set Menu res ID to 1051.
6. Set resource Info (CMD-I).
7. In the Info window, set the ID to 1050 and the name to "Main Menu".
8. Close all the windows back to the MemoPadMain.rsrc type picker.

2.1.1.2 Creating a Menu

To create a menu (MENU) resource to hang on the menu bar, follow these steps (see the "Resource Cookbook" for details):

1. Create a new resource (CMD-K).
2. Enter MENU as the resource type and click OK.
3. In the editor window, set the menu title to "Options".
4. Set resource Info (CMD-I).
5. Set the ID to 1051 and the name to "Options Menu".
6. Close the Info window.

7. If you get a dialog asking to update the menu ID to 1051, click OK.

2.1.1.3 Creating a Menu Item

Populate the menu you just created with a menu item. Add one item and name it "Get Info"; follow these steps:

1. In the MENU editor window, choose Resource>Create New Item (CMD-K).
2. Set the menu item text to "Get Info" and Cmd-Key to 'I'.

Set a resource ID for the Get Info menu item; follow these steps:

1. Choose MENU>Edit Menu & MDEF ID.
2. Set the Menu ID to 100. Subsequent menu items increment this number. Leave the MDEF ID as 0. Then, click OK.
3. Close the editor window and the MENU resource picker.
4. Save the resource file.
5. Quit ResEdit (CMD-Q).

2.1.2 Examining the Code

2.1.2.1 Adding #define Macros for New Resources

The MemoPadRsc.h file contains #define macros for the menu bar and the Info command. Open MemoPadRsc.h from CodeWarrior and take a look.

Note: You may wonder why the programmer assigned these particular IDs in this application. At this point, they may not make sense. However, as you get deeper into this application and see the development of the MemoPadRsc.h file, the reasoning behind this scheme should become evident.

2.1.2.2 Adding a Menu Event Handler

The file MemoPad.c in the :Src folder includes code changes to incorporate a menu bar with an Options menu. From CodeWarrior; open MemoPad.c.

Take a look at the changes to MemoPad 1. Notice that the programmer has added "P2" to some comments. These are entries made for Phase 2. **Tip**: To locate changes for this phase, search with "P2" as the target string.

Near the top of the file, Phase 2 adds a global variable, CurrentMenu, to keep track of the currently active menu by way of a pointer. **Note**: Each form can have its own menu bar and set of menus.

The StartApplication routine now contains a call to a new function, SetCurrentMenu. This function sets up the menus for the initial (Main) form, based on the mainMenu resource.

The SetCurrentMenu routine follows. It first checks for the need to dispose of an existing menu. (Note the CurrentMenu was initialized to NULL.) Then, the new menu is initialized by a call to MenuInit.

Phase 2 adds a test for a menuEvent to the MainFormHandleEvent routine. The identity of the menu item is included in the event structure. In this case, there is only one menu item, so you can presume it's the Get Info item. Choosing this item triggers a series of events, culminating in the appearance of the Info dialog. These events are:

- Call the MenuEraseStatus function to clear the menu command prompt from the display.
- Initialize the Info form.
- Display the Info form.

The last two are described in detail in a later section. Take a look at the EventLoop routine. For Phase 2, the programmer has inserted a call to MenuHandleEvent to process menu events. Note that this new event handler is placed immediately after SysHandleEvent and before MainFormHandleEvent. The system still gets the first shot at each event, but the menu takes priority over the application's private handling of the form.

2.1.3 Adding a Get Info Form with an OK Button

With a menu structure in place, the programmer needs to add an OK button to the dialog. Generally, we'll use a separate resource file to contain each form for the application. Start by building a new re-

source file named MemoPadInfo.rsrc to contain the new form and new resources:

- A title for the Info dialog
- An OK button to dismiss the dialog
- Labels to act as the text of the dialog: “Memo Pad” and “Version 1.0”
- A string to hold the Help text for the dialog
- A form bitmap for the company logo
- A PICT graphic to hold the actual logo

2.1.3.1 Creating a Resources Document for the Get Info Form

Create a new resources document, MemoPadInfo.rsrc to contain the resources for the Info dialog. Follow these steps:

1. With ResEdit running, choose File>New and enter MemoPadInfo.rsrc as the name.
2. In the file dialog, set the path to Working Folder:Rsc and click New. A type picker named MemoPadInfo.rsrc appears on the screen.
3. With the new resource file active, build the info form’s resources.

2.1.3.2 Creating a New Form

With ResEdit running and the MemoPadInfo.rsrc resource file open, create a form (tFRM) resource for the memo pad. Fill in the form fields as follows:

1. Set the dimensions to 2, 46, 156, 112.
2. Leave all the radio buttons set to True and set the Form ID to 1100.
3. Set resource Info; choose Resource>Get Resource Info (CMD-I).
4. Set the ID to 1100 and the name to “Memo Pad Info.”
5. Close the Info window, the tFRM Memo Pad Info editor window, and the tFRM resource picker.

2.1.3.3 Creating a Title Resource

With the type picker open, create a title (tTTL) resource for the Info dialog. Fill in the fields as follows:

1. Set the Title to “Info”.
2. Set resource Info; choose Resource>Get Resource Info (CMD-I).
3. Set the ID to 1101 and the name to “Memo Info”.
4. Close all windows back to the type picker.

2.1.3.4 Creating a Button Resource

Create a button (tBTN) resource to dismiss the Info dialog. Fill in the fields as follows:

1. Set the Button ID to 1105.
2. Set the button’s position to 60, 95, 37, 12.
3. Leave all the radio buttons set to True.
4. Set the font to 0 and the label to “OK”.
5. Set resource Info; choose Resource>Get Resource Info (CMD-I).
6. Set the ID to 1105 and the name to “OK”.
7. Close all windows back to the type picker.

2.1.3.5 Creating a Label Resource

Create a label (tLBL) resource to display the application name in the Info dialog. Fill in the fields as follows:

1. Set the Label ID to 1103.
2. Set the left, top coordinates to 50, 28.
3. Set the font to 1 and the Text to “Memo Pad”.
4. Set resource Info; choose Resource>Get Resource Info (CMD-I).
5. Set the ID to 1103 and the name to “Appl Name”.
6. Close all windows back to the type picker.

2.1.3.6 Creating a Second Label Resource

Create another label (tLBL) resource to display the application version in the Info dialog. Fill in the fields as follows:

1. Set the Label ID to 1104.
2. Set the left, top to 50, 39.
3. Set the font to 1 and the Text to “Version 1.0”.

4. Set resource Info; choose Resource>Get Resource Info (CMD-I).
5. Set the ID to 1104 and the name to "Version #".
6. Close all windows back to the type picker.

2.1.3.7 Creating a String Resource

Create a new string (tSTR) resource to contain the Help text for the Info dialog. Fill in the fields as follows:

1. Set the String to "The Info dialog provides information about the application such as the development company and version number."

   Note: For the purposes of this project, you can safely ignore the Data field.
2. Set resource Info (CMD-I).
3. Set the ID to 1107 and the name to "Help: Info form".
4. Close all windows back to the type picker.

2.1.3.8 Creating a Form Bitmap Resource

Create a form bit map (tFBM) resource to display and position the development company's logo. Fill in the fields as follows:

1. Set X and Y coordinates to 5 and 25, respectively and the BitmapRsc ID to 1110.
2. Set resource Info; choose Resource>Get Resource Info (CMD-I).
3. Set the ID of the form Bitmap Resource to 1102 and the name to "Company Logo".
4. Close all windows back to the type picker.

2.1.3.9 Creating a PICT Resource

To put a graphic into a PICT resource you need some kind of graphic editor, like MacPaint. (You could pre-load a graphic into the Macintosh Scrapbook, making it easy to access when programming.

In the steps below, you copy your PICT image from a graphic editor into the PICT resource.

Begin by putting the company logo on the Clipboard; follow these steps:

1. Open your graphic editor and open the file named Company Logo. It is located in the MemoPad 02:Rsc folder.
2. Use the lasso tool to select only the graphic.
3. Copy the selected graphic onto the Clipboard.

Create a PICT resource to display the company's logo. Complete the resource as follows:

1. Choose Resource>Create New Resource (CMD-K).
2. In the Select New Type dialog, enter PICT and click OK.
3. With the PICT window open, paste the logo in from the Clipboard.
4. Set resource Info; choose Resource>Get Resource Info (CMD-I).
5. Set the ID to 1110 and the name to "Company Logo". This ID establishes the connection to the form bit map (tFBM) resource.
6. Close all windows back to the type picker.

2.1.3.10 Adding Resources to a Form

Now that you have constructed a number of resources for your form, add them to the object array in the form resource; follow these steps:

1. From the type picker window, open the tFRM resource picker.
2. Open the Memo Pad Info form (ID 1100). Add the new UI components to the form's collection of objects.
3. To establish the connection to the help string resource, set the Help Rsc ID to 1107.
4. Set the Default Button ID to 1105.
5. Below the label "Number of objects" click the item labeled "1) ******"
6. Insert a new field (CMD-K).
7. Fill in the information for the title resource. Set Obj ID to 1101 and Obj Type to tTTL.
8. Click the item labeled "2) ******"
9. Insert a another field (CMD-K).

10. Fill in the information for the form bit map resource. Set Obj ID to 1102 and Obj Type to tFBM.
11. Click the item labeled "3) ******"
12. Insert a another field (CMD-K).
13. Fill in the information for the first label resource. Set Obj ID to 1103 and Obj Type to tLBL.
14. Click the item labeled "4) ******"
15. Insert a another field (CMD-K).
16. Fill in the information for the second label resource. Set Obj ID to 1104 and Obj Type to tLBL.
17. Click the item labeled "5) ******"
18. Insert a another field (CMD-K).
19. Fill in the information for the button resource. Set Obj ID to 1105 and Obj Type to tBTN.

That's all the array work for this session. Go ahead and wrap it up with ResEdit:

1. Close all windows back to the type picker.
2. Save the resource file.

2.1.4 Examining the Code

2.1.4.1 Adding a #define Macro for the InfoForm Resource

The system handles the processing of the Info form pretty much on an automatic basis. For this reason, the app doesn't need to refer to the ID's of its components. To identify your resource to your application code, the programmer added a #define to the MemoPadRsc.h file for the infoForm resource ID. Switch to CodeWarrior and take a look at the file. Notice that the infoForm macro has a value of 1100, the same as its ID. Remember, you can search the comments for "P2" in MemoPadRsc.h and in MemoPadRsc.c (see below).

The programmer also includes the Info form resources (MemoPadInfo.rsrc) and the system's Help resources (:::Resources:Help Dialog, a.k.a. Tips) in the MemoPadRsc.c file.

2.1.4.2 Adding a Handler for the Info Dialog with Do Dialog

The file MemoPad.c includes code changes required to use the Info dialog box. Open MemoPad.c from CodeWarrior and take a look at the changes since Phase 1.

The MainFormHandleEvent routine now contains a test for a menuEvent, as discussed above. In review, here's what happens:

- A call to the MenuEraseStatus function clears the menu command prompt, so the Info dialog can be displayed.
- A call to FrmInitForm loads the Info form into memory.
- A call to FrmDoDialog displays the Info form. FrmDoDialog takes care of displaying the Info form and handling all the user events, while the form is displayed.
- When the user taps the OK button on the Info form, FrmDoDialog restores the display to its appearance before the Info form was displayed.
- When FrmDoDialog returns, the FrmDeleteForm function deletes the no longer needed Info form from memory.

2.2 Building and Exercising the Application

To build and exercise the phase 2 application, follow these steps:

1. With CodeWarrior running, open MemoPadμ.
2. Add MemoPadInfo.rsrc to the project under the Resources section. Drag the Working Folder:Rsc:MemoPadInfo.rsrc file icon to the project window and drop it in under Resources.
3. Check the Project menu to make sure that the debugger is disabled before running the application. It should show the item Enable Debugger. If the item says Disable Debugger, choose it.
4. Build the application (CMD-M).
5. Launch the application (CMD-R). The Pilot Simulator starts and displays the Pilot with your memo pad application in it.

Try out the menu system:

1. Display the Options menu and the Get Info menu item; tap the Menu button.

2. Display the Info form; tap the Get Info item.
3. Display Help for the form on the Info form, tap the Tips icon to the right of the form's title.
4. Dismiss the Tips form; tap the Done button.
5. Dismiss the Info form; tap the OK button.
6. Quit the application; tap the main form's Exit button.

Adding a Button, an Edit Form, and Navigation

Overview

Phase 3 adds a New button, a full-screen edit form, and some navigation between the edit form and the main form. See Figures 3.1 and 3.2. To set up the appropriate files for this phase, follow these steps:

1. If MemoPadMain.rsrc and MemoPadInfo.rsrc do not already exist in Working Folder:Rsc from Phase 2, then copy them from the MemoPad 02:Rsc folder.
2. Copy MemoPad.c, MemoPadRsc.c, and MemoPadRsc.h from the MemoPad 03:Src folder to Working Folder:Src.
3. If MemoPad.μ does not exist in Working Folder from Phase 2, copy it from MemoPad 02.

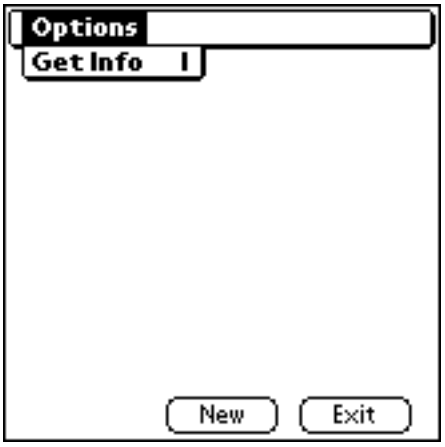

Figure 3.1 The Phase 3 Memo Pad with its New button.

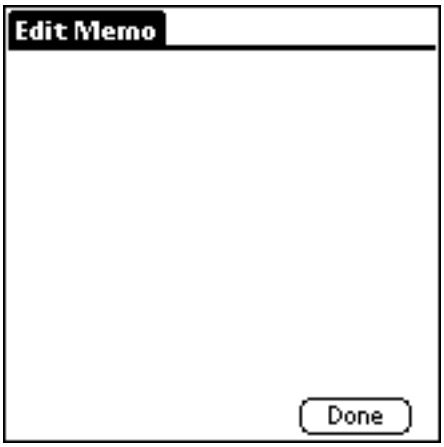

Figure 3.2 The Phase 3 Memo Pad with its edit form.

3.1 Adding a New Button to the Main Form

This section shows you how to:

- Add a button resource to the main form
- Connect the menu to the main form, so as make menu processing more automatic.

It also describes the code for activating that resource.

3.1.1 Creating a "New" Button

This section shows you how to create a "New" button for the main form. This button, when tapped, displays the edit form.

To begin, open the main form resource file in ResEdit:

1. Open the Working Folder:Rsc folder.
2. Open the MemoPadMain.rsrc type picker.

Create a button (tBTN) resource, and fill in the fields as follows:

1. Set the Button ID to 1003.
2. Set the button's dimensions to 60, 146, 40, 12.
   Leave all the radio buttons set to True.

3. Set the Font to 0 and the Label to “New”.
4. Set resource Info (CMD-I).
5. Set the ID to 1003 and the Name to “New”.
6. Close all windows back to the type picker.

3.1.2 Adding the New Button Resource to the Main Form

Now that you have a new button, add it to its parent form; follow these steps:

1. From the MemoPadMain.rsrc type picker window, open the tFRM resource.
2. In the resource picker window, open the Memo Pad resource for the main form, Memo Pad Main.
3. Attach the menu to the main form. Set the value of Menu Rsc ID to 1050 to refer to the main menu bar.
4. Add the New button to the main form. Click the item labeled “3) ******” and create a new field.
5. Set Obj ID to 1003 and Obj Type to tBTN.
6. Close all windows back to the type picker.
7. Save the resource file (CMD-S).

3.1.3 Examining the Code

3.1.3.1 Adding a #define Macro for the “New” Button Resource

To identify a resource to the application code, add a #define to the MemoPadRsc.h file for the New button resource ID. Open MemoPadRsc.h from CodeWarrior. Notice that the mainNewButton macro has a value of 1003, the same as its ID.

3.1.3.2 Adding Event-Handler Code for New Button

The file MemoPad.c includes changes required to use the New button. Open MemoPad.c from CodeWarrior and take a look at the changes from MemoPad 2.

The MainFormHandleEvent routine now contains a switch statement to pass control to one of several cases based on event type. The

case for ctlSelectEvent now includes a test to determine which button was pressed—Exit or New.

The test matches a button's resource ID to the ID of the pressed button contained in the event structure. When the New button is pressed, a call to FrmGotoForm with the ID of the Edit form loads and displays the Edit form, after closing the Main form.

FrmGotoForm does this by adding frmCloseEvent, frmLoadEvent and frmOpenEvent events to the event queue. The application retrieves these events later from the EventLoop routine.

The MainFormHandleEvent routine also handles the frmOpenEvent event for the Main form. When the Edit form's Done button is pressed, the Main form displays. When the frmOpenEvent event arrives, the Main form has been loaded and activated and remains only to be drawn on the display.

The handled variable is set to True to indicate that subsequent event handlers should not process these events.

3.2 Creating an Edit Form with a Title and a Done Button

This section shows you how to create an Edit form and its resources.

3.2.1 Creating the Edit Form and Its Resources

The first version of the Edit form is similar to the first version of the main form. It contains only a title and a button. Clicking the button causes an exit from the form and a return to the main form.

3.2.1.1 Borrowing from MemoPad 01

Create the Edit form resource from the previous version of the Main form; follow these steps:

1. Switch to the Finder and duplicate MemoPad 01:Rsc:MemoPadMain.rsrc (Press CMD-D).
2. Drag the copy to Working Folder:Rsc and rename it to MemoPadEdit.rsrc.

Adapt the form to MemoPad 03 by revising the Edit form's form, title, and button resources.

3.2.1.2 Creating the Edit Form

Modify the form resource to create the Edit form; follow these steps:

1. Continue in ResEdit and double-click on the new file MemoPadEdit.rsrc to open.
2. From the MemoPadEdit.rsrc type picker window, open the tFRM resource picker.
3. Open the Memo Pad Main item.
4. Change the Form ID to 1200.
5. Change the ID for the tTTL object to 1201.
6. Change the ID for the tBTN object to 1202.
7. Set resource Info (CMD-I).
8. Set the ID to 1200 and the name to "Memo Pad Edit".
9. Close all windows back to the type picker.

3.2.1.3 Creating the Edit Form Title

Modify the title resource; follow these steps:

1. From the MemoPadEdit.rsrc type picker window, open the tTTL resource picker.
2. Open the Memo Pad item.
3. Change the title to: "Edit Memo".
4. Set resource Info (CMD-I).
5. Change the ID to 1201 and the name to "Edit Memo".
6. Close all windows back to the type picker.

3.2.1.4 Creating the Done Button

Modify the button resource; follow these steps:

1. From the MemoPadEdit.rsrc type picker window, open the tBTN resource picker.
2. Double-click the only entry in the list.
3. Change the Button ID to 1202.
4. Change the Label to "Done".
5. Set resource Info (CMD-I).

6. Change the ID to 1202.
7. Change the name to “Done”.
8. Close all windows back to the type picker.
9. Save the resource file and quit ResEdit.

3.2.2 Examining the Code

3.2.2.1 Adding #define Macros for New Resources

To identify the resource to the application code, the programmer added two #define macros to the MemoPadRsc.h file. Switch to CodeWarrior and examine the file. Notice that the values for the editForm (1200) and editDoneButton (1202) macros are the same as their IDs.

3.2.2.2 Adding Edit Form Handling to the Code

The file MemoPad.c includes changes required to use the Edit form. Use CodeWarrior to open MemoPad.c and take a look at the changes.

The EditFormHandleEvent routine is new. It handles two events that require special processing for the Edit form: clicking the Done button and opening a form. This example has two events to be handled.

If the event type is ctlSelectEvent, it means the user has tapped a button. The identity of the button is included in the event structure and is verified to be the Done button. This button causes the Edit form to close and the Main form to display.

A call to FrmGotoForm with the ID of the Main form closes the Edit form while loading and displaying the Main form. FrmGotoForm does this by adding frmCloseEvent, frmLoadEvent and frmOpenEvent events to the event queue. The application retrieves these later in the EventLoop routine.

The EditFormHandleEvent also handles the frmOpenEvent for the Edit form. This event is generated when the Main form’s New button is pressed. When the frmOpenEvent arrives, the Edit form has already been loaded and activated. All that remains is for it to be drawn on the display.

For both these events, the EditFormHandleEvent routine returns a value of true to indicate that subsequent event handlers should not process the event.

3.2.2.3 Other Changes to the Application Code

Several other changes have been made to the example code for this phase. The addition of FrmGotoForm to go between forms accompanies the Edit form. This introduces the frmLoadEvent and a new mechanism for dispatching events to each form's event handler.

In the PilotMain routine, a call to FrmGotoForm with CurrentView as an argument follows StartApplication. Inspection of the StartApplication routine shows the previous calls to InitForm, SetActiveForm and DrawForm are no longer used. Instead a global variable, CurrentView, is set to the ID of the (soon-to-be) current form.

As mentioned above, FrmGotoForm adds frmCloseEvent, frmLoadEvent and frmOpenEvent events to the event queue. These events will be retrieved later by the application in the EventLoop routine.

Inspection of the EventLoop routines shows that the calls to ApplicationHandleEvent and FrmDispatchEvent replace calls to MainFormHandleEvent and FrmHandleEvent. Take a look at the new ApplicationHandleEvent routine, just above EventLoop, to understand what's happening.

The ApplicationHandleEvent routine processes the frmLoadEvent event. The structure of this event contains the ID of the form to be loaded. InitForm loads the form, and SetActiveForm activates it.

Finally, the system is informed which application routine will process events for that form. For the Main form, this is the MainFormHandleEvent routine. For the Edit form, it is the EditFormHandleEvent routine. FrmDispatchEvent (called in the EventLoop routine) calls these routines for each event processed, while the form is active.

Another change is the setup of the main menu. The resource for the Main form was modified above to include setting the menu rsc ID to the ID of the menu bar resource. This lets the default form-event processing routine, FrmHandleEvent (called internally by FrmDispatchEvent) handle setup and teardown of the menu for a form. As

a result, SetCurrentMenu is no longer required, because FrmSetActiveForm handles this task automatically.

3.3 Building and Exercising the Application

With resources and code in place for Phase 3, build the project and exercise it in the Pilot Simulator; follow these steps:

1. With CodeWarrior running, open MemoPad.μ, and check the Project menu to make sure that the debugger is disabled before running the application. It should show the item Enable Debugger. If the item says Disable Debugger, choose it.
2. Add MemoPadEdit.rsrc to the project in the Resources section. Drag the Working Folder:Rsc:MemeoPadEdit.rsrc icon to the project window and drop it in under Resources.
3. Choose Project>Remove Binaries.
4. Build the application (CMD-M).
5. Launch the application (CMD-R). The Pilot Simulator displays your memo pad application.
6. Testing the system:
   a. Get the Edit form; tap the Main form's New button.
   b. Tap the Menu button. Notice that there is no menu for this form.
   c. To return to the Main form, tap the Done button.

3.4 Using the Event Trace Window

One of the Pilot Simulator's debugging features is its Event Trace window see Figure 3.3. Select Window>Event Trace to activate the Event Trace window. The Event Trace window will display all events running through the main event queue, along with some of their relevant data. Exercise the application and observe the event flow. Since the Palm OS UI is largely single-threaded, this event flow gives an excellent view of the state of the system at any time.

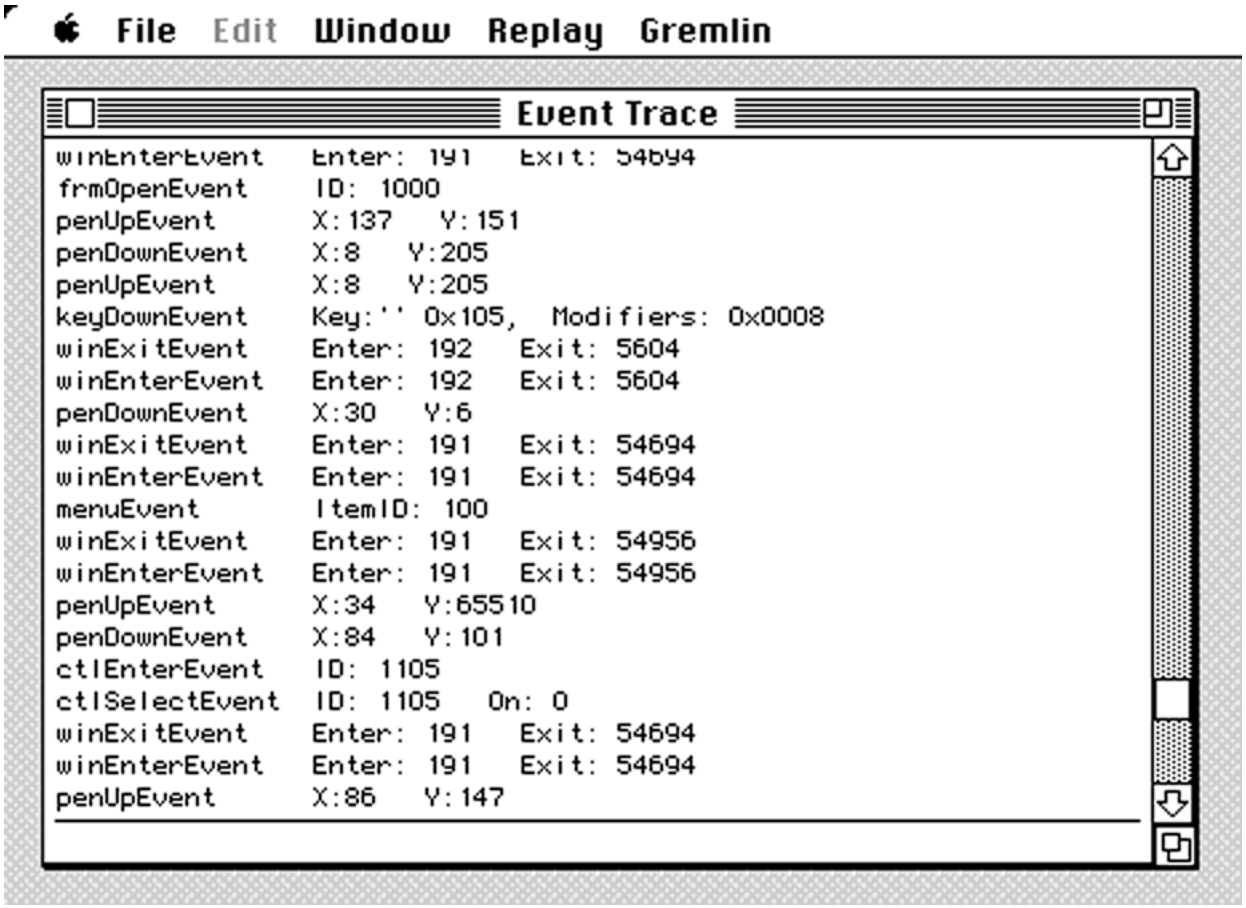

Figure 3.3 The Event Trace window.

Adding a Text Field and Menu Commands

Overview

Phase 4 adds a text field, an Edit menu, and menu commands to make field interaction possible. See Figures 4.1, 4.2, and 4.3. You'll learn how to set up basic edit commands and field navigation commands. To set up the appropriate files for this phase, follow these steps:

1. If MemoPadMain.rsrc, MemoPadInfo.rsrc, and MemoPadEdit.rsrc do not already exist in Working Folder:Rsc from Phase 3, then copy them from the MemoPad 03:Rsc folder.
2. Copy MemoPad.c, MemoPadRsc.c, and MemoPadRsc.h from the MemoPad 04:Src folder to Working Folder:Src.
3. If MemoPad.μ does not exist in Working Folder from Phase 3, copy it from MemoPad 03.

Figure 4.1 Phase 4 Memo Pad with text entered in Edit area.0

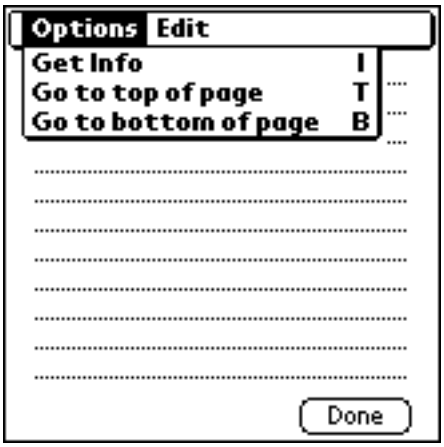

Figure 4.2 Phase 4 Memo Pad with additional Options menu items.

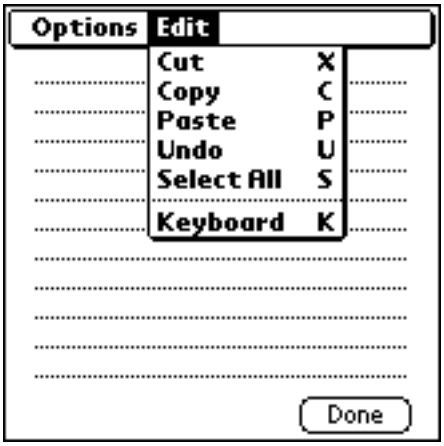

Figure 4.3 Phase 4 Memo Pad with Edit menu items.

4.1 Adding a Large Text Field to the Edit Form

Editing memos requires a text field. In this section, you'll create one. You'll also create a Graffiti UI object to handle the Graffiti shift symbol.

To begin, open the edit resource file in ResEdit, follow these steps:

1. Open the Working Folder:Rsc folder.

2. Open MemoPadEdit.rsrc.

The Edit form needs a text field (tFLD) resource. Follow these steps:

1. Create a tFLD resource.
2. Set the Field ID to 1203.
3. Set the field's position to 10, 17, 140, 130.
4. Set Single Line to False. Leave all the radio buttons set to True.
5. Set Max chars to 1028 and Font to 0.
6. Set resource Info (CMD-I).
7. Set the ID to 1203 and the name to "Edit Field".
8. Close all windows back to the type picker.

4.2 Adding a Resource for the Graffiti Shift UI

The Graffiti Shift Indicator (tGSI) specifies the position where the Graffiti Shift symbol (for punctuation, symbol, uppercase shift, and uppercase lock) appears in the form window. Follow these steps:

1. Create a tGSI resource for the Edit form.
2. Put the resource in the lower left of the Pilot display. Set the x position to 5 and the y position to 146.
3. Set resource Info (CMD-I).
4. Set the ID to 1204 and the Name to "Graffiti Shift".
5. Close all windows back to the type picker.

4.2.1 Adding the Field and Graffiti UI Resources to the Edit Form

Now that you have a new field, add it to its parent form; follow these steps:

1. From the type picker, open the tFRM resource.
2. From the tFRM resource picker, open Memo Pad Edit.
3. Set the Menu Rsc ID to 1250.
4. Click the item labeled "3) ******".
5. Create a new field (CMD-K).

6. Insert the edit field; set ID to 1203 and Type to tFLD.
7. Click the item labeled "4) ******".
8. Create a new field (CMD-K).
9. Insert the Graffiti Shift field; set ID to 1204 and Type to tGSI.
10. Close all windows back to the type picker.
11. Save the resource file (CMD-S).

4.2.2 Examining the Code

4.2.2.1 Adding #define Macros for New Resources

To identify resources to the application code, the programmer added two #define macros to the MemoPadRsc.h file. Open MemoPadRsc.h from CodeWarrior. Notice that the values for the editField (1203) and editDoneButton (1202) macros are the same as their IDs.

NOTE: You needn't add the 1204 ID for the Graffiti UI to the header file. The reason is that the application doesn't interact directly with this resource, so you don't need to tell your code it's here. The same is true of some other resources, like titles and strings.

4.2.2.2 Revising a Handler for the Text Field

The file MemoPad.c in the MemoPad 04 folder includes changes required to use the text field in the Edit form. Open MemoPad.c from CodeWarrior and take a look at the changes from MemoPad 03. These changes support the new resources in Phase 4.

Within the EditFormHandleEvent routine, the programmer made an addition to the frmOpenEvent case. After the form is drawn on the screen, the focus for text input needs to be set to the text field. A call to FrmSetFocus, passing the index of the text field, sets the focus.

4.3 Adding Resources for the Edit Menu and Options Menu

This section shows you how to:

- Add a menu to the Edit form.
- Populate the menu with commands.
- Add two commands to the Options menu.

The steps to do all this are:

- Copy resources from Main and modify them for Edit.
- Add commands to the Edit menu.
- Set Edit menu IDs.
- Add commands to the Options menu.
- Add the Edit menu on the menu bar.

4.3.1 Copying Resources from Main to modify them for Edit

Initially, the Edit form menu looks similar to that of the Main form. It contains an Options menu with a Get Info item that displays the Info form.

Leverage your earlier work by creating the Edit form's menu resource from the menu of the Main form. Follow these steps:

1. Switch to ResEdit.
2. Open the file MemoPadEdit.rsrc.
3. Open the file MemoPadMain.rsrc.
4. In MemoPadMain.rsrc, select the MENU and MBAR resource types and copy the selection.
5. Select MemoPadEdit.rsrc and paste in the two resources. Both menu resources appear in the target window.
6. Close the MemoPadMain.rsrc window.

Edit the resource copies you just made to suit the Edit menu; follow these steps:

1. Select the MemoPadEdit.rsrc type picker and open the MENU resource.
2. Select the entry for Options Menu.
3. Set resource Info (CMD-I).
4. Set the ID to 1251.
5. Close the Info window. If you get a dialog asking to update the menuID, click Cancel.
6. Close the MENU resource picker.
7. Open the MBAR resource picker and the Main Menu resource.
8. Select the first entry and change the Menu res ID to 1251.
9. Set resource Info (CMD-I).
10. Set the ID to 1250 and the name to “Edit Form Menu”.
11. Close all windows back to the type picker.

4.3.2 Adding Commands to the Edit Menu

Now that you have a menu for the Edit form, populate it with menu items, commands. Begin by creating and populating an Edit menu follow these steps:

1. In the MenuPadEdit.rsrc window, open the MENU resource picker.
2. Create a new resource (CMD-K).
3. Set the menu title to “Edit”. Do NOT close the window.
4. Create a new item (CMD-K).
5. Set the menu item text to “Cut” and the Cmd-Key to X. Press Enter.
6. Set the menu item text to “Copy” and the Cmd-Key to C. Press Enter.
7. Set the menu item text to “Paste” and the Cmd-Key to P. Press Enter.
8. Set the menu item text to “Undo” and the Cmd-Key to U. Press Enter.

9. Set the menu item text to “Select All” and the Cmd-Key to S. Press Enter.
10. Click the radio button for separator line. Press Enter.
11. Click the radio button for Text.
12. Set the menu item text to “Keyboard” and set the Cmd-Key to K. The Keyboard command displays the on-screen keyboard.

4.3.3 Setting Edit Menu IDs

With the Edit menu completed, set its IDs; follow these steps:

1. Set resource Info; choose Resource>Get Resource Info (CMD-I).
2. Set the ID to 1252 and the name to “Edit Menu”.
3. Close the Info window.
4. Cancel the warning dialog.
5. Choose MENU>Edit Menu & MDEF ID....
6. Set the Menu ID to 200 and click OK.
7. Close the MENU “Edit Menu” window.

4.3.4 Adding Commands to the Options Menu

Continuing in ResEdit, add two navigational commands to the Options menu:

1. Select the Option Menus entry in the MENU resource picker and open it.
2. Create a new item (CMD-K).
3. Set the menu item text to “Go to top of page” and the CMD-Key to T. Press Enter.
4. Set the menu item text to “Go to bottom of page” and the CMD-Key to B.
5. Close the MENU “Options Menu” editor window and the MENU resource picker.

4.3.5 Adding the Edit Menu to the Menu Bar

Continuing in ResEdit, add two navigational commands to the Options menu:

1. Open the MBAR resource picker and the Edit Form Menu resource.
2. Click the item labeled "2) ******".
3. Insert a new field (CMD-K).
4. In the new entry, set the Menu res ID to 1252 to add the Edit menu to the menu bar.
5. Close the MBAR "Edit Form Menu" editor window and the MBAR resource picker.
6. Save the resource file.

4.3.6 Examining the Code for the New Edit Menu and Option Resources

With new resources in place, take a look at the code changes needed to activate the resources.

4.3.6.1 Adding #define Macros for New Resources

To identify the resources to the application code, the programmer added a #define macro to the MemoPadRsc.h file for the Edit menu and the new menu command resource IDs. Switch to CodeWarrior and take a look at the file.

4.3.6.2 Adding an Event Handler for the Edit Menu

The file MemoPad.c includes changes required to use the new menus in the Edit form. Use CodeWarrior to open MemoPad.c and take a look at the changes from MemoPad 03.

Within the EditFormHandleEvent routine, the programmer has added a new case for the menuEvent. First, MenuEraseStatus clears the menu command prompt from the display. Then, EditDoMenuCommand executes the menu command (based on the command ID as extracted from the event structure).

4.3.6.3 Adding a Menu Item Handler for the Edit Form Menu

EditDoMenuCommand is a new routine located just above EditFormHandleEvent. It has responsibility for handling all the menu commands for the Edit form. The getInfo command displays the Info form using the same actions as were put in place earlier for the menuEvent in the MainFormHandleEvent routine.

4.3.6.4 Adding Handlers for Menu Commands

The goToTop command moves the location of the text insertion point to the beginning of the text field. It calls GetFocusObjectPtr to retrieve a pointer to the field that currently has the focus for text insertion. If a field currently has the focus, FldSetInsPtPosition is called to set the insertion point to position zero, the beginning of text. This will cause the text to scroll in the field, when necessary.

The goToBottom command moves the insertion point to the end of the text field. If a field currently has the focus, the routine calls FldGetTextLength to get the length of the text and FldSetInsPtPosition to set the insertion point to the end of text. This also will cause the text to scroll in the field, when necessary.

The cutCmd command places a copy of the currently selected text on the system clipboard and removes the text from the field. FldCut handles all this and the necessary display updates.

The copyCmd command places a copy of the currently selected text on the system clipboard and doesn't alter the text in the field. FldCopy handles these chores.

The pasteCmd command inserts a copy of the text currently on the system clipboard into the text field at the location of the insertion point. FldPaste handles all this as well as the necessary display updates.

The undoCmd command reverses the last text change. FldUndo handles this including the necessary display updates.

The selectAllCmd command selects all the text in the field. FldSetSelection is called to select all the text from position zero to the last character for the field.

The keyboardCmd command displays the on-screen keyboard. SysKeyboardDialog handles display of the keyboard and all its interac-

tions, including updating text with the changes made through the keyboard.

GetFocusObjectPtr is a new routine located just above EditDoMenuCommand. It finds the text field in the current form that has the focus, if any, and returns a pointer to the field structure.

4.4 Building and Exercising the Application

To build the MemoPad project, follow these steps:

1. With CodeWarrior running, check the Project menu to make sure that the debugger is disabled before running the application. It should show the item Enable Debugger.
2. Choose project/remove binaries.
3. Build the application (CMD-M).
4. Launch the application (CMD-R). The Pilot Simulator starts and displays the Pilot with your memo pad application in it.

To try out the system, follow these steps:

1. Display the edit form; tap the main form's New button. Note the flashing insertion point bar at the beginning of the text field.
2. Tap the Menu button. You get a menu bar with the Options and Edit menus.
3. Display the Info form; tap the Get Info item, which displays the Info form.
4. Close the Info form.
5. Type or write with Graffiti some words into the text field, like "Palm OS is the best handheld system ever!".

With some text in place, experiment with moving the insertion point; follow these steps:

1. Notice that you can change the location of the insertion point by tapping on the displayed text.
2. Tap the Menu button and select the Go to top of page item. Note that the insertion point bar jumps to the beginning of

the text. Similarly, the Go to bottom of page item moves the insertion point to the end of text.

Give the edit commands a try; follow these steps:

1. Try the command stroke. (The command stroke is a diagonal slash from lower left to upper right in the Graffiti area.)
2. Select some of the text by clicking and dragging over it.
3. Tap the menu button and choose the Edit>Copy item. The selected text is copied to the clipboard.
4. Choose the Paste item. This inserts text from the clipboard into the text field at the insertion point, thus creating a second copy of the text in the field.
5. Select all the text in the field; choose the Select All item.
6. Choose the Cut item to copy selected text to the clipboard and remove it from the displayed text.
7. Undo the Cut operation; select the Undo item.

Exercise the on-screen keyboard; follow these steps:

1. Display the on-screen keyboard; tap the Edit>Keyboard item and use it to add some characters to the text.
2. Dismiss the keyboard; tap the keyboard's Done button.

Take a look at how your application handles larger amounts of text; follow these steps:

1. Enter several lines of text, even if you just copy the initial text and paste it several times. Note that the text field automatically word wraps at the end of a line and scrolls the lines, when the number of lines is greater than the lines on the display.
2. Return to the main form, tap the Done button.
3. Quit the application, tap the main form's Exit button.

Storing and Retrieving Text in a Database

Overview

In this phase, you add some data management to the application. By the end of this phase, you'll be able to store and retrieve text data entered into the text field of the Edit form. To do this, you create a Palm OS database and within it, a record to store the data. This phase keeps it simple and works with a single record. Figure 5.1 shows changes to the main forms. To set up the appropriate files for this phase, follow these steps:

1. If MemoPadMain.rsrc, MemoPadInfo.rsrc, and MemoPadEdit.rsrc do not already exist in Working Folder:Rsc from Phase 4, then copy them from the MemoPad 04:Rsc folder.
2. Copy MemoPad.c, MemoPadRsc.c, and MemoPadRsc.h from the MemoPad 05:Src folder to Working Folder:Src.
3. If MemoPad.µ does not exist in Working Folder from Phase 4, copy it from MemoPad 04.

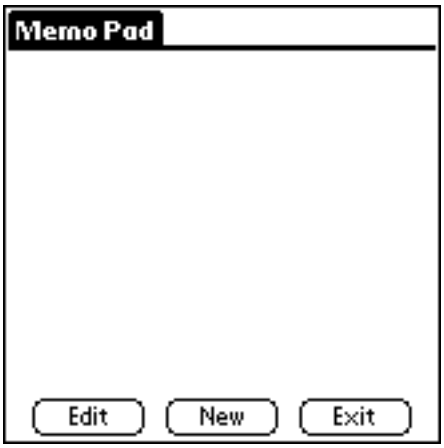

Figure 5.1 The Phase 5 Memo Pad

5.1 Adding Text Storage in a Database Record

This section describes the basics of Palm OS databases and describes three key functions for running a database.

5.1.1 Database Basics

The first step is to create a new database for this application. Normally, each Palm OS application has its own database for storing its records. The database resides in the Palm OS device's non-volatile storage. Each database has a name, a creator, and a type. The name can be up to 31 characters in length. Because it is the primary identifier, it must be unique within the system. The creator is a four-character value that must be either all uppercase or mixed case. An all-lowercase name is illegal, because the Palm OS system software reserves all-lowercase names for itself. The type is also a four-character value that must be either all-uppercase or mixed case.

5.1.2 Database Create, Open, And Close Functions

Take a look at the code for Phase 5 to see how storage is handled. From CodeWarrior open MemoPad.μ and select MemoPad.c.

There are two key functions in Phase 5 that handle starting and stopping an application, each uses a database:

- StartApplication
- StopApplication

StartApplication has been revised to create and open a database for this application. Here's how:

- StartApplication looks for an existing application database. The DmFindDatabase function returns an ID for the database that matches the requested name. If the database doesn't exist, a call to the DmCreateDatabase function creates it. If the create fails, there is a call to ErrFatalDisplay, and the application exits.
- Normally, the create succeeds, so FindDatabase is called again to get an ID for the database. The ID is used to open the database for both read and write access.
- The handle to the opened database is stored in a new global variable named MemoPadDB. Another new global variable, CurrentRecord, is initialized to indicate that no database record is being processed. This variable stores the index of the current database record.

Here's how the application database works. The application's database is kept open while the application is running. PilotMain calls it just prior to the application's exit. StopApplication is a new routine. StopApplication cleans up the state of the application's data before exit. Finally, DmCloseDatabase closes the database.

5.2 Revising the Handler for the Done Button

With the picture of stopping and starting in mind, you may wonder what happens in between these programmatic bookends. Take a look at these functions:

- EditFormHandleEvent
- EditSaveData
- DmNewRecord

and how they use these functions:

  – MemHandleLock

- DmWrite
- MemHandleUnLock
- DmReleaseRecord

5.2.1 EditFormHandleEvent

The programmer has modified EditFormHandleEvent, so that it saves the Edit form's text-field data to the application's database, when the form is done. If the CtlSelectEvent case finds that the Done button has been selected, it does the following:

- It gets a pointer to the edit field's data structure
- It passes the pointer to a new routine named EditSaveData.
- It resets the CurrentRecord global variable to indicate that there is no longer a current record.

5.2.2 EditSaveData

The EditSaveData function's job is to write text-field data to the first record of the application's database. Here's how it works:

- It gets a pointer to the text field.
- It tests the pointer for data in the field to ensure that there is an allocated string for the text and that the string is not empty.
- Given some data in the field, it calls DmNewRecord to create a new record in the database and return a pointer to the newly created record. The new record size equals the byte count for text-field data and an additional zero terminator character.
- A call to MemHandleLock locks the handle, thus locking the record's location in memory, before directly manipulating the data in the record. MemHandleLock also returns a pointer to the record in memory.
- A call to DmWrite writes the data from the text field to the record, starting at location zero in the record.
- Once the record's data has been set, MemHandleUnlock unlocks the record, so that the memory system may move the record's location, if required by fragmentation problems.
- DmReleaseRecord releases the record to the system's database manager. When the record was initially created it was

marked internally by the system as being busy, meaning in use by the application. Releasing the record lets the database manager regain control of the record from the application. In the call to DmReleaseRecord the application also indicates to the database manager that the record contains "dirty" data, i.e., data that has been changed since it was last released.

5.3 Adding an Edit Button to the Main Form

Adding a button is a question of setting up the right resources and a bit of coding. Begin with the resources.

5.3.1 Creating the Edit Button Resource

To begin, open the resource file for the main form in ResEdit:

1. Open the Working Folder:Rsc folder.
2. Open the MemoPadMain.rsrc type picker.

Add an Edit button to the main form, follow these steps:

1. Open the tBTN resource picker.
2. Create a new button resource for the Edit button (CMD-K).
3. Set the Button ID to 1004.
4. Set the dimensions to 10, 146, 40, and 12.
5. Leave all the radio buttons set to True
6. Set the Font to 0 and the Label to "Edit".
7. Set resource Info (CMD-I).
8. Set the ID to 1004 and the name to "Edit".
9. Close all the windows down to the type picker.

5.3.2 Adding the Edit Button to the Main Form

Continue by incorporating the new button into the main form; follow these steps:

1. Open the tFRM resource picker and, from within it, the Memo Pad Main form resource.
2. Click the item labeled "4) *****".

3. Insert a new field (CMD-K).
4. Fill in the information for the Edit button. Set the ID to 1004 and the Type to tBTN.
5. Close the tFRM form editor window and the tFRM resource picker.
6. Save the resource file.

5.2.3 Examining the Code

5.2.3.1 Adding a #define Macro for the Edit Button

To identify a resource to the application code, the programmer added a #define macro to the MemoPadRsc.h file for the Edit button resource ID. Open MemoPadRsc.h from CodeWarrior and take a look at the changes from MemoPad 4. Notice that the mainEditButton macro is assigned the ID value 1004, to match the resource.

5.2.3.2 Adding a Handler for the Edit Button

The programmer modified the MainFormHandleEvent routine to handle the new Edit button by adding a new case for the mainEditButton. To tell the Edit form that the user selected the Edit button, the CurrentRecord global variable is set to the index of the record to be edited. In this example, it is always the first record in the database, index value zero. Calling FrmGotoForm initiates the transition to the edit form.

5.2.3.3 Revising the Handler for the New Button

The MainFormHandleEvent routine's handling of the New button has been modified. Now, it tells the Edit form that this is a new record by setting the CurrentRecord global to a special value of noRecordsSelected. Then, a call to FrmGotoForm initiates transition to the edit form.

5.3 Adding Retrieval of Text from a Database Record

Adding text retrieval is a question of revising the Edit Form Handler.

5.3.1 Revising the Edit Form Handler

In the Phase 5 version, EditFormHandleEvent reads the first record's data from the application's database into the edit form's text field when the form is opened. The case for the openFormEvent event now includes getting a pointer to the edit field's data structure. It passes this pointer to a new routine named EditRetrieveData.

The EditRetrieveData routine reads the first record of the application's database into the text field. The routine first checks to see if there is a current record by checking the value of the CurrentRecord global variable. It also checks to see if the database contains any records. If the New button is tapped on the main form, the value of CurrentRecord is noRecordSelected. In this case, no attempt is made to read from the database, and the field retains its initialized value, an empty string.

If a record is present, these actions take place:

- A call to DmGetRecord gets a handle for the first record in the database. (Remember that CurrentRecord was set to zero in the code-handling mainEditButton).
- Before the data in the record can be directly manipulated, the handle must be locked to pin down the record's location in memory. MemHandleLock does this and returns a pointer to the record in memory.
- A new block of memory is allocated to contain a copy of the data for the field of the form. MemHandleNew is called with a size determined by the size of the data in the record, plus one for a zero terminator on the string. The function returns a handle to the new memory block.
- MemHandleLock is called again to pin down the new memory block. Now the data from the first database record is copied to the new memory block via StrCopy. The new memory block is then unlocked, so that it is again under the control of the memory system. The new memory block, containing a copy of the record data can now be provided to the field. FldSetTextHandle takes the handle of the new memory block and sets the value of the field to the text contained in the new memory block.

- Now that the record's data has been retrieved, it is unlocked so that the memory system may move its location, if required by fragmentation problems.
- The record is released to the system's database manager. When the record was initially created, it was marked internally by the system as being busy, meaning in use by the application. Releasing the record lets the database manager regain control of the record from the application. In the call to DmReleaseRecord, the application also indicates to the database manager that the record has not been changed since the call to DmGetRecord.

5.4 Building and Exercising the Application

To build the MemoPad project, follow these steps:

1. With CodeWarrior running, check the Project menu to make sure that the debugger is disabled before running the application. It should show the item Enable Debugger.
2. Build the application (CMD-M).
3. Launch the MemoPad application (CMD-R). The Pilot Simulator starts and displays the Pilot with your memo pad application in it.

Write, save, and review your work; follow these steps:

1. Display the edit form; tap the main form's New button.
2. Enter some words into the text field, like "Palm OS is the best handheld system ever!". **Note**: To change the location of the insertion point, tap the displayed text.
3. Return to the main form; tap the Done button.
4. Cause the edit form to re-appear; tap the edit button. Note that the previously entered text is displayed again.
5. Return to the main form; tap the Done button.
6. Display the edit form; tap the main form's New button. Note that the text field is empty.
7. Enter new text into the text field, like "I'm going to create the best application ever for Palm OS!".
8. Return to the main form; tap the Done button.

9. Cause the edit form to re-appear, tap the edit button. Note that the new text is displayed again.
10. Return to the main form; tap the Done button.

The next activity shows you how to interact with the Console, using its commands; follow these steps:

1. Tap the Pilot Simulator's Window>Console menu item to display the Console window.
2. Select the Console window and press Enter (or CMD-Return) to move the insertion point to a new line.
3. Type the command "opened" and press Enter. A list of currently open databases appears. The first entry is named "MemoPadDB" and is the application's database.

   Note the value in the accessP column of the list. This value is known as the access pointer of the database. The two TRES databases are resource databases. One contains application resources and the other contains system resources as specified in MemoPadRsc.c. Graffiti contains macros and appears automatically.
4. Next, examine information about the records in the Memo Pad database. Type the command "listrecords", add a space, and type the value of accessP for Memo Pad's database. For example:

   ```
   listrecords 00DAB484
   ```

   It is often more convenient and less error prone to use copy / paste on access pointers.

   A list of the current records appears, including the index, some of the data, and other information. Note that our current application's record list (as shown in Figure 5.2) contains duplicates of each record. This is because the application in its current state creates a copy of a record for editing, and never deletes the copy. This will be taken care of in later phases.
5. Quit the application, tap the main form's Exit button.

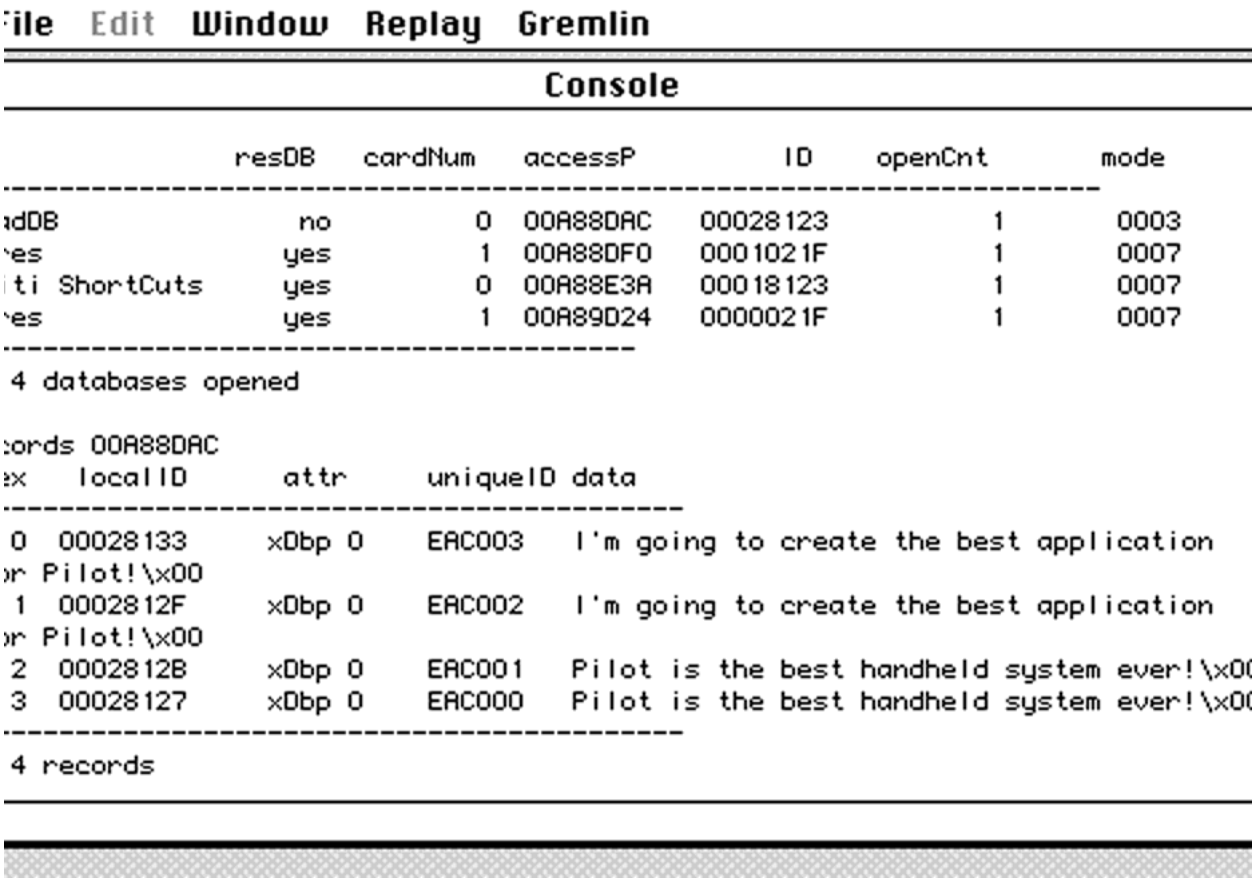

Figure 5.2 The Console

Editing a Data Record in Place

Overview

Phase 6 introduces a technique for working with data records known as edit in place. Phase 6 makes no changes to resources.

To set up the appropriate files for this phase, follow these steps:

1. If MemoPadMain.rsrc, MemoPadInfo.rsrc, and MemoPadEdit.rsrc do not already exist in Working Folder:Rsc from Phase 5, then copy them from the MemoPad 05:Rsc folder.
2. Copy MemoPad.c, MemoPadRsc.c, and MemoPadRsc.h from the MemoPad 06:Src folder to Working Folder:Src.
3. If MemoPad.µ does not exist in Working Folder from Phase 5, copy it from MemoPad 05.

6.1 About Edit-in-Place

In the previous example (Phase 5), the data is copied from the long-term storage to a temporary buffer in dynamic memory (RAM) for editing.

The user interacts with the data in the buffer by means of the user interface. The buffer absorbs and contains these modifications. When the display and editing are complete, the updated contents of the buffer are written back to long term storage. Because of the inefficiencies of editing directly in long-term storage on most systems, you need to create the extra copy and move the data back and forth.

The Palm OS system allows for a more efficient mechanism called “edit-in-place”. Long-term storage is simply a section of the system’s battery-backed dynamic memory. There is no requirement to move data from one area of dynamic memory to another for presentation and editing in the user interface.

This means the user, through the user interface, can edit the data in a database record directly, while it is still in the database. You don't need to put a temporary copy of the data in the system memory area. This has the advantage of minimizing usage of the limited amount of system memory, while avoiding the performance overhead of extra memory allocations and data copies.

6.2 Adding a Handler for Edit in Place

The file MemoPad.c includes code changes required to set up an edit-in-place mechanism. From CodeWarrior, open MemoPad.c and take a look at the changes from MemoPad 5.

Go to the mainNewButton case of MainFormHandleEvent function. To create a new record in the database, MainNewButton calls a new routine CreateRecord. If CreateRecord fails, the edit form is not displayed. Since all editing is done directly in a database record, a new record must be created before editing can begin. (The previous example relied on the system's form processing to provide an initial buffer in dynamic memory for the field's data.)

The call to CreateRecord could have been made after the edit form gains control. However, if there is a problem from CreateRecord; the edit form is never started, and there is no switch to and from the edit form.

CreateRecord calls DmNewRecord to create a new record, and an empty first record in the application's database. The new record has a default size of newMemoSize. The data in the record needs to be initialized to an empty string. A call MemHandleLock gets a pointer to the record data. A single zero byte is written to the first position in the record.

DmWrite returns an error code that is checked by a call to ErrFatalDisplayIf. This routine checks the value of error and, if it is nonzero, displays an alert message box containing the specified error message. This alert does not let the program continue and should only be used in cases where the error condition is truly fatal. This is similar to the way StartApplication works (See Phase 5). If its call to DmCreateDatabase fails, it calls ErrFatalDisplay.

If there is no error from writing the record, it is unlocked. Then, the record is released to the system's database manager, letting the database manager regain control of the record from the application. In the call to DmReleaseRecord, the application also indicates to the database manager that the record contains "dirty" data, i.e., data has been changed. Finally, the global variable CurrentRecord gets the index of the first record.

The EditRetrieveData routine has now been simplified. Once the current record has been retrieved by a call to DmGetRecord the handle of the record is passed to the field by a call to FldSetTextHandle. This directs the system's field processing to access and modify the field data by referencing the handle of the record. That is, presentation and editing will be done directly in the database record. (The code from MemoPad 5 that has been replaced is included for reference.)

The programmer has also simplified the EditSaveData routine. A call to FldGetTextPtr gets a pointer to the field's data and stores it in the variable called text. A call to FldSetTextHandle clears the handle to the field data. This clears the reference to the field data in the field data structure pointed to by the variable fld. Even though the handle has been cleared, the field data is still available via the variable text.

It is important to clear the handle reference in the field data structure. The reason is that when the edit form is closed, the form is disposed of and cleared from memory. Part of that process frees all memory blocks obtained by the objects within the form. If the field data structure contains a handle for the field's data, the memory referenced by that handle is freed. In many cases, this is appropriate and desirable. However, in this example, the handle is referencing data within the database record, so you don't want it freed.

Lastly, the database record must be released back to the datbabase manager's control with the dirty flag set. (The code from MemoPad 5 that has been replaced is included for reference.)

Note: If you fail to clear the handle reference and there is data in the field, you risk data loss, a crash, and an error message.

Adding a List to Display and Select a Record

Overview

This phase adds a scrolling list to the main form. See Figure 7.1. To work this phase, follow these steps:

If MemoPadMain.rsrc, MemoPadInfo.rsrc, and MemoPadEdit.rsrc do not already exist in Working Folder:Rsc from Phase 6, then copy them from the MemoPad 06:Rsc folder.

1. Copy MemoPad.c, MemoPadRsc.c, and MemoPadRsc.h from the MemoPad 07:Src folder to Working Folder:Src.
2. If MemoPad.μ does not exist in Working Folder from Phase 6, copy it from MemoPad 06.

Figure 7.1 The Phase 7 Memo Pad with a scrolling list.

The scrolling list displays an item for each record in the application database. Each item shows a line width (approximately the first 30-or-so characters) of text from the record. If there are more items than fit within the list's boundaries, the list scrolls. When you tap an item, the edit form displays the entire text of the corresponding record in the text field.

NOTE: The display list shown in Figure 7.1 and described in this phase will be replaced later with a table in the main form. Using the list provides a way to show how to use a list and postpones the more challenging topic of tables.

7.1 Adding Record Display List to the Main Form

In this section, you do three things to the main form:

- Add a list resource
- Remove the Edit button.
- Remove the Exit button.

7.1.1 Adding a List Resource

Add a list (tLST) resource to the main form.

To begin, open the resource file for the main form in ResEdit:

1. Open the Working Folder:Rsc folder.
2. Open the MemoPadMain.rsrc type picker.

Add a list resource to the main form; follow these steps:

1. Create a new list resource (CMD-K).
2. Type tLST for the resource type and click OK.
3. Set the List ID to 1005.
4. Set the list's dimensions to 10, 17, and 140.
5. Set Font to 0 and visible items to 11.
6. Set resource Info (CMD-I).
7. Set the ID to 1005 and the name to "Memo list".
8. Close all the windows down to the type picker.

7.1.2 Removing Buttons

This section shows you how to remove the Exit button from the main form. Just as it takes two activities to add an object to a form, it

takes two to remove one: clear the object from the resource picker and remove it from the form's object array. With ResEdit open, remove the Exit and Edit buttons; follow these steps:

1. Open the tBTN resource.
2. Select the Exit item.
3. Remove the Exit button, choose Edit>Clear.
4. Select the Edit item.
5. Remove the Edit button, choose Edit>Clear.
6. Close the tBTN resource picker.

With the buttons gone from the resource list, remove them from the main form; follow these steps:

1. Open the tFRM resource and, within it, the "Memo Pad Main" resource.
2. Click the item labeled "2) *****". (It holds the ID 1002, the Exit button.)
3. Remove this reference from the form; choose Edit>Clear.
4. Click the item labeled "3) *****". (It holds the ID 1004, the Edit button.)
5. Remove this reference from the form; choose Edit>Clear.

7.1.3 Adding a List Resource

To add the list resource to the main form, follow these steps:

1. Include the list resource; click the item labeled "3) *****" and insert a new field (CMD-K).
2. Fill in the information for the list; set ID to 1005 and Type to tLST.
3. Close all the windows down to the type picker.
4. Save the resource file.

7.1.4 Revising Code for a List of Records

The adding or removing of a resource necessitates changes to code—to header files and to handlers. In particular, code needs to be revised for the removal of the Exit button and for the inclusion of

list handling. This section walks you through these changes. Its subsections include:

- Revising #define macros in the header file
- Revising a handler to set up a list at frmOpenEvent
- Revising a handler to free the list at frmCloseEvent
- Revising a handler for the Edit button
- Add a handler to set the CurrentRecord for the lstSelectEvent
- Removing the handler for the Exit Button

7.1.4.1 Revising #define Macros

From CodeWarrior, open the MemoPadRsc.h file. Note three changes to the previous version:

- To identify the list resource to the application code, the programmer added a #define macro to the MemoPadRsc.h file: mainList (1005)
- Because the Exit Button resource is gone, so is its macro mainExitButton (1002).
- Likewise, because the Edit Button resource is gone, so is its macro mainEditButton (1004).

7.1.4.2 Revising the Handler to Set up a List of Records

The goal is to have a list of database records appear when the main form opens. Continuing in CodeWarrior, open the MemoPad.c file. The programmer has modified the MainFormHandleEvent routine to set up the list of records in the database. When the frmOpenEvent case activates, it calls a new routine, MainFormInit, to set up the list and draw the main form. In the MainFormInit routine, the call to DmNumRecords shows no records in the database. No list setup is needed, but the form is drawn.

When a database contains records, you must set up the list, so that each database record is represented by one item. Each item consists of the first few characters of the record. The width of the list's display rectangle limits the number of characters displayed for an item.

The LstSetListChoices function passes items to be displayed in the list to the system's list processing code. The items in the list must be organized and formatted appropriately for that function. See Figure

7.2. That means an array of pointers to zero-terminated text strings, where each string will become an item in the list.

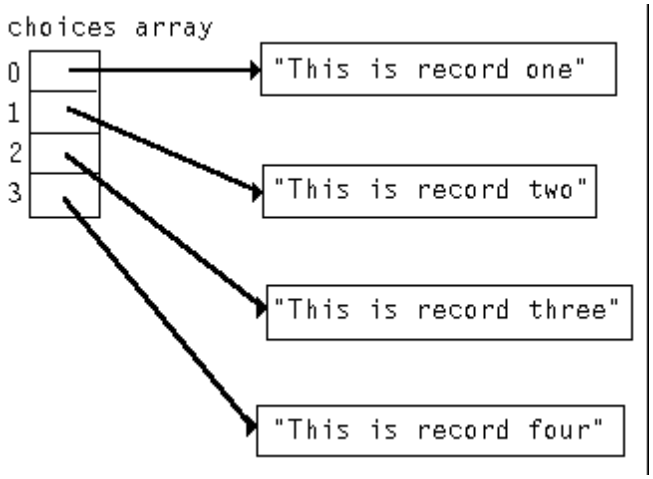

Figure 7.2 An array of pointers to strings.

Constructing this array structure from the database records is an interesting exercise in memory allocation and pointer management. However, there is a system function that relieves some of the complexity. The function SysFormPointerArrayToStrings constructs an array of pointers given a memory block that contains the required strings in a packed form (what the loop builds for you).

NOTE: Packed strings are strings concatenated without additional separators. For example, the packed string “record one” “record two” “record three” “record four” packs the four separate strings:
“record one”
“record two”
“record three”
“record four”

The strings in the block are placed one directly after the other with a zero byte terminating each string. Given such a collection of packed strings, SysFormPointerArrayToStrings creates and sets up an array of pointers appropriate for LstSetListChoices. The loop within the MainFormInit routine creates a collection of packed strings from the database records.

To determine the number of characters you can display, do the following:

1. Get a pointer to the list.
2. Get the width (in pixels) of the list's rectangle from the list data structure. This width is decreased by a couple of pixels to ensure there is no overlap of the item text on the frame of the list.

A new memory block, sized to one character, is allocated and its handle saved in the global variable ChoicesHandle. The handle must be global, because it must be freed later in another routine, when the main form is closed. The memory block is locked in place and its initial value set to a zero to indicate an empty string.

Here's the process for each record in the database:

- The record is retrieved and locked in place.
- Determine the length of the record text that fits within the list bounds. If the record consists of multiple lines, the list item should only display the first line. To find the first linefeed character in the string, Call StrChr. If one is found, textLen is set to the length of text up to the linefeed. Otherwise, textLen is set to the total length of the string.
- A call to the function FntCharsInWidth determines how many characters of the string can fit within the bounds of the list rectangle.
- Once the length of the string is known, the choices buffer (an array of packed strings) is resized larger to contain a copy of the text from the record.
- The data is copied from the database record to the current location in the choices buffer, as indicated by choicesOffset.
- A zero terminator goes at the end of the copied text
- The choicesOffset variable gets updated to the location for the next record text.
- The record is unlocked and released back to the system's control.

When all the records in the database have been processed, the SysFormPointerArrayToStrings function is called to create an array of pointers from the packed strings in the choices memory block. The function returns a handle for the newly allocated memory block containing the array of pointers. This handle is stored in a global

variable, ChoicesPtrsHandle. The handle must be global, because it will be freed when the main form is closed. The memory block is locked in place and passed to LstSetListChoices to establish the collection of strings as the items to be displayed in the list. The list is displayed as part of the form by the call to FrmDrawForm.

7.1.4.3 Revising a Main Form Handler to Free the List at Form Close

To free the list at frmCloseEvent, the MainFormHandleEvent routine includes processing for the frmCloseEvent case. Before the form is closed, the memory blocks allocated for the list items must be freed. An if statement tests ChoicesHandle to ensure that the list is populated. If it is, both the Choices and the ChoicesPtrs memory blocks are freed. Note that the handled variable is not set to True for this event. In fact, the system's form-handling code needs to process this event as part of its own form cleanup.

7.1.4.4 Removing the Handler for the Exit and Edit Buttons

The user doesn't explicitly exit a Palm OS application. Rather, the user simply launches another application. When this happens, the system automatically adds an appStopEvent to the currently executing application's event queue. When running a Palm OS application within the Pilot Simulator, this is simulated by choosing File>Quit from the Pilot Simulator's menu or by pressing Escape.

7.1.4.5 Adding a Handler to Set the CurrentRecord for the lstSelectEvent

The MainFormHandleEvent routine now includes processing for the lstSelectEvent case. This event indicates that the user tapped an item in the list. Event processing gets the index of the list item and saves it in CurrentRecord. Then, a call to FrmGotoForm switches to the edit form.

7.2 Adding Multiple Records to the Database

Previous steps have almost completed this. Tapping the New button on the main form calls the CreateRecord routine. It always inserts a new record as the first record before the previous first record in the database. The list setup in MainFormInit has the ability to handle

multiple records in a database. Tapping an item that represents a record in the list sets the CurrentRecord global variable to the index of the selected record. When the edit form processes the frmOpenEvent, a call to RetrieveRecord retrieves the record indicated by CurrentRecord.

One small cleanup task remains—eliminating empty database records. Such records cause problems for the list and incur unnecessary overhead in the database. This version of the EditSaveData routine automatically deletes a record from the database, in either of two cases:

- If you create a new record and don't enter any data in it.
- If you completely delete the text of an existing record.

After retrieving the pointer to the text data, a test determines if text is present. If the current database record is empty, a call to DmRemoveRecord removes it from the database.

7.3 Building and Exercising the Application

With resources and code in place for Memo Pad 7, build the project and exercise it in the Pilot Simulator; follow these steps:

1. With CodeWarrior running, check the Project menu to make sure that the debugger is disabled before running the application. It should show the item Enable Debugger.
2. Build the application (CMD-M).
3. Launch the application (CMD-R). The Pilot Simulator starts and displays the Pilot with your memo pad application in it.

With the application running; follow these steps:

1. Display the edit form; tap the main form's New button.
2. Enter several words into the text field.
3. Return to the main form; tap the Done button. The main form now displays a list with a single item at the top already selected.
4. Display the edit form again; tap the item in the list. Note that the previously entered text is displayed again. Return to the main form; tap the Done button.

5. Tap the New button on the main form to cause the edit form to be displayed.
6. Type, or write with Graffiti several words into the text field such as "I'm going to create the best application ever for Palm OS!".
7. Return to the main form; tap the Done button. The main form now displays a list with two items. The more recent item appears as the first item of the list and is selected. It is truncated to fit within the horizontal bounds of the list rectangle. Tapping either item displays its text in the edit form.
8. Tap either item and, in the edit form, delete all the text of the record.
9. Return to the main form; tap the Done button. The Palm OS system has removed the item from the list and from the database.
10. Quit the application; choose the Pilot Simulator File>Quit item.

Adding Display List Items as Required

Overview

Phase 8 describes an alternative to packed strings for delivering display items, “display-as-required”. This phase also shows you how to use Gremlins to exercise your application. To work this phase, follow these steps:

1. If MemoPadMain.rsrc, MemoPadInfo.rsrc, and MemoPadEdit.rsrc do not already exist in Working Folder:Rsc from Phase 7, then copy them from the MemoPad 07:Rsc folder.
2. Copy MemoPad.c, MemoPadRsc.c, and MemoPadRsc.h from the MemoPad 08:Src folder to Working Folder:Src.
3. If MemoPad.µ does not exist in Working Folder from Phase 7, copy it from MemoPad 07.

8.1 The Display-As-Required Approach

The previous example (Phase 7) displays a list of all items in the database. To do this, the application makes a copy of the first portion of text in each record and makes that copy a list item. As a result, each record in the database is represented by an item in the list. But there are problems with this approach. If the database grows to contain a large number of records in the list, a collection of packed strings, could exhaust available dynamic memory. In addition, this approach takes extra time and storage to create list items for records that are not displayed in the list, unless it is scrolled.

The Palm OS system offers a more efficient approach. The key is that you don’t have to hand an array of packed strings to the list processing code. Instead, the application can provide each item only when it’s time to display that item. The system’s list processing code can ask the application to draw each list item, when it is required.

8.2 Examining the Code

The file MemoPad.c incorporates changes for drawing the list items as required (No changes have been made to the resources from the Phase 7). From CodeWarrior, open MemoPadRsc.h and take a look at the changes from MemoPad 7.

In the MainFormInit routine, LstSetListChoices is now called with a null pointer for the array of list choices. Next the function LstSetDrawFunction is called and passed the address of the application's MainFormListDrawItem routine. This tells the system that this application routine has responsibility for drawing the items in the list. LstDrawList calls this routine once for each item that is displayed in the list. Note this includes only those items that are currently displayed in the list, not every item that might be shown. Scrolling the list calls MainFormListDrawItem once for each item that scrolls into view.

The new MainFormListDrawItem routine is passed the item number, which in this example is interpreted as the record number and the display bounds within which the list item should be drawn. (The third argument, itemsText, is not used in the example.) The task is to draw the data for this item and draw it within the given bounds.

The record with index itemNum is retrieved from the database and locked down. The code still calculates the length of the text to be drawn as it did in the previous example. However, now the bounds argument is used to constrain the width. WinDrawChars is called to draw the characters on the display starting at the top-left corner of the bounds. Once the item is added to the list, the record is unlocked and released back to the system.

8.3 Building and Exercising the Application

With resources and code in place, build the project and exercise it in the Pilot Simulator; follow these steps:

1. With CodeWarrior running, check the Project menu to make sure that the debugger is disabled before running the application. It should show the item Enable Debugger.
2. Build the application (CMD-M).

3. Launch the application (CMD-R). The Pilot Simulator starts and displays the Pilot with your memo pad application in it.

With the application running, use the Gremlins feature to exercise it. This feature exercises the app by generating a series of random input events, including

- Pen taps on buttons
- Pen taps on other screen gadgets
- Entering text into fields.

Gremlins is an effective way to test an application for failures due to unanticipated user input. Gremlins continues beating up the system, until you choose the Gremlin>Stop item. Go ahead and run Gremlins; follow these steps:

1. From the Pilot Simulator menu, choose Gremlin>New.
2. When you're done with the Gremlins, choose Gremlin>Stop.

Adding a Page Menu to the Edit Form

Overview

This phase adds a page menu with new page and delete page commands. To work this phase, follow these steps:

1. If MemoPadMain.rsrc, MemoPadInfo.rsrc, and MemoPadEdit.rsrc do not already exist in Working Folder:Rsc from Phase 8, then copy them from the MemoPad 08:Rsc folder.
2. Copy MemoPad.c, MemoPadRsc.c, and MemoPadRsc.h from the MemoPad 09:Src folder to Working Folder:Src.
3. If MemoPad.μ does not exist in Working Folder from Phase 8, copy it from MemoPad 08.

9.1 Adding a Page Menu

The edit form of the application currently has two menus. The Page Menu will be the third. It will have two items: New Page and Delete Page.

Selecting New Page will save any changes to the currently displayed record and clear the form for a new record. If the currently displayed record is empty, there is no effect.

Selecting Delete Page will display a dialog box to confirm the deletion of the current record. After confirmation, the current record will be deleted and the edit form will be closed.

9.1.1 Adding Menu Items for New Page and Delete Page

This section shows you how to:

- Add a new menu: Page

- Add two items to the Page menu: New Page and Delete Page
- Include these items in the MBAR container.

To begin, open the resource file for the edit form in ResEdit:

1. Open the Working Folder:Rsc folder.
2. Open the MemoPadEdit.rsrc type picker.

The first task is to add resources for deleting a page; follow these steps:

1. Open the MENU resource picker; double-click the MENU resource item.
2. With the type picker selected, create a new resource (CMD-K).
3. Set the Title to “Page”. Do not close this window.

With a menu created, populate it with menu items; follow these steps:

1. Create a new item for the menu; create a new item (CMD-K).
2. Set the menu item Text to “New Page” and set CMD-Key to N.
3. Add another new item for the menu; press Enter.
4. Set the menu item Text to “Delete Page” and set CMD-Key to D.

Now, finish up; follow these steps:

1. Set resource Info (CMD-I).
2. Set the ID to 1253 and the name to “Page Menu”.
3. Close the Info window.
4. Cancel the dialog that asks to update the menuID to 1253.
5. Choose MENU>Edit Menu & MDEF ID.
6. Set Menu ID to 300 and click OK.
7. Close the MENU “Page Menu” editor window and the MENU resource picker.

Add the new menu to the menu bar; follow these steps:

1. Open the MBAR resource picker.
2. Open the Edit Form Menu item.
3. Click the item labeled “3) *****”.
4. Insert a new field (CMD-K).
5. Set the Menu res ID to 1253.
6. Close the MBAR “Edit Form Menu” editor window and the MBARs resource picker.
7. Save the resource file.

9.1.2 Adding a Resource for a Delete Memo Alert

In this section, you build an alert dialog that appears when the user deletes a memo; follow these steps:

1. With the type picker still open, create a new Talt resource (CMD-K). This opens the alert editor.
2. Set the Alert type to 2.
3. Set # Buttons to 2.
4. Set Default Button to 1.
5. Set the Title to “Delete Memo”.
6. Set the Message to “Do you wish to permanently remove this page?”.

Create the buttons for the alert dialog; follow these steps:

1. Click the item labeled “1) *****”.
2. Insert a new field (CMD-K).
3. Set Button Text to “OK”.
4. Click the item labeled “2) *****”.
5. Insert a new field (CMD-K).
6. Set Button Text to “Cancel”.

With the resource constructed, set resource Info; follow these steps:

1. Choose Resource>Get Resource Info (CMD-I).
2. Set the ID to 1210 and the name to “Delete Alert”.

Go ahead and wrap it up.

1. Close the Info window, the Talt “Delete Alert” editor window, and the Talt resource picker.
2. Save the resource file.

9.1.3 Examining the Code

9.1.3.1 Adding #define Macros for New Resources

To identify resources to the application code, the programmer added four #define macros to the MemoPadRsc.h file. Switch to CodeWarrior and take a look at the file. Notice that the programmer has entered values for the new resources:

- editDeleteAlert (1210)
- editDeleteOK (0)
- newPageCmd (300)
- deletePageCmd (301)

9.1.3.2 Adding Handlers for the New Commands

The file MemoPad.c includes changes required for the new menu commands. Open MemoPad.c from CodeWarrior and take a look at the changes from MemoPad 08. The programmer has added two new cases, newPageCmd and deletePageCmd, to the EditDoMenuCommand routine.

9.1.3.3 How the New Page Menu Command Works

To create a new memo page, the newPageCmd follows these steps:

- EditSaveData saves the current data in the edit field to the database.
- FldDelete clears the field of any existing text in preparation for a new record.
- CreateRecord creates a new first record in the database and sets CurrentRecord accordingly.

- EditRetrieveData sets up the new record with the field for an edit-in-place operation.

9.1.3.4 How the Delete Page Menu Command Works

To delete a memo page, the deletePageCmd case follows these steps:

- To ensure it's the user's intention to delete a page, deletePageCmd displays an alert dialog.
- FrmAlert gets the ID of the delete alert. FrmAlert displays the alert dialog and handles all the user interaction until the OK or Cancel button is tapped.
- When the user taps a button, FrmAlert erases the alert dialog and restores the display.
- FrmAlert returns a value that indicates which button was tapped.
- If the OK button is tapped, any text in the field is deleted.
- EditSaveData saves the current field data. Because the field is empty, this causes the record to be removed from the database.
- CurrentRecord is set to indicate that there is no current database record.
- FrmGotoForm replaces the edit form with the main form.

9.2 Building and Exercising the Application

With resources and code in place, build the project and exercise it in the Pilot Simulator; follow these steps:

1. With CodeWarrior running, check the Project menu to make sure that the debugger is disabled before running the application. It should show the item Enable Debugger.
2. Build the application (CMD-M).
3. Launch the application (CMD-R). The Pilot Simulator starts and displays the Pilot with your memo pad application in it.

With the application running; follow these steps:

1. Tap the New button.
2. Type some text into the new page.

3. Tap the Done button.
4. Add another page by choosing Page>New Page from the menus.
5. Add some text into the new page.
6. Tap the Done button.
7. From the List View, select one of the records.
8. Delete the page; choose Page>Delete Page and respond to the Delete alert dialog.
9. Quit the application; choose the Pilot Simulator File>Quit item.

Adding a Details Dialog and the Secret Record Attribute

Overview

In Phase 10, you add a Details button and dialog plus a secret record attribute. See Figure 10-1.

To work this phase, follow these steps:

1. If MemoPadMain.rsrc, MemoPadInfo.rsrc, and MemoPadEdit.rsrc do not already exist in Working Folder:Rsc from Phase 9, then copy them from the MemoPad 09:Rsc folder.
2. Copy MemoPad.c, MemoPadRsc.c, and MemoPadRsc.h from the MemoPad 10:Src folder to Working Folder:Src.
3. If MemoPad.μ does not exist in Working Folder from Phase 9, copy it from MemoPad 09.

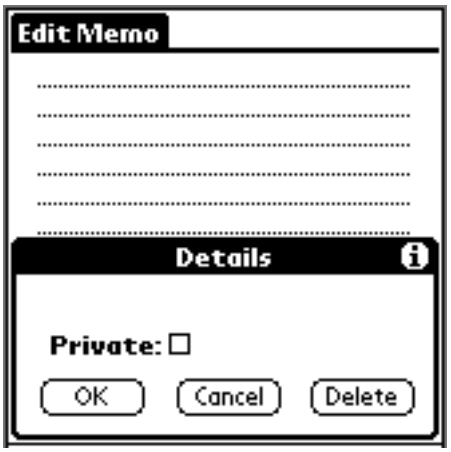

Figure 10.1 Phase 10 Memo Pad with Details dialog.

10.1 Adding a Details Dialog to the Edit Form

In this section, you add a Details button to the edit form and a Details dialog to the project.

10.1.1 Adding a Resource for Details Button on Main Form

This section shows you how to create a Details button on the main form.

To begin, from ResEdit, open the resource file for the edit form:

1. Open the Working Folder:Rsc folder.
2. Open the MemoPadEdit.rsrc type picker.

Add the button; follow these steps:

1. Within the type picker, open the tBTN resource type.
2. Create a new tBTN resource (CMD-K).
3. Set the Button ID to 1205.
4. Set the dimensions to: 60, 146, 40, and 12.
5. Set Font to 0 (zero) and the Label to “Details”.
6. Set the resource Info (CMD-I).
7. Set the ID to 1205 and the name to “Details”.
8. Close all the windows down to the type picker.

Add the Details button to the edit form; follow these steps:

1. Open the tFRM resource.
2. Open the Memo Pad Edit form (ID 1200)
3. Click the item labeled “5) ******”.
4. Insert a new field (CMD-K).
5. Insert the edit field; set ID to 1205 and Type to tBTN.
6. Close all the windows down to the type picker.
7. Save the resource file.

10.1.2 Creating a Details Resource File

In this section, you create a new resource file to contain the resources for the Details dialog. The new resource file is named MemoPadDetails.rsrc; follow these steps:

1. With ResEdit open, choose File>New.
2. In the new resource dialog, navigate to the Working Folder:Rsc folder.
3. In the New File Name box, type MemoPadDetails.rsrc.
4. Click the New button.

10.1.2.1 Creating a Details Form

Create a form to contain the dialog's various resources; follow these steps:

1. With the type picker selected, create a new resource (CMD-K).
2. Create a form resource for the Details form; in the Select New Type dialog, enter tFRM and click OK.
3. Set the dimensions to 2, 86, 156, 72.
4. Set the Form ID to 1300.
5. Set the resource Info (CMD-I).
6. Set the ID to 1300 and the Name to "Memo Pad Details".
7. Close all the windows down to the type picker. ResEdit may generate a warning about an "Invalid Rectangle" while closing the tFRM edit window; ignore it.

10.1.2.2 Creating a Details Title

Create a title (a tTTL resource) for the Details dialog; follow these steps:

1. From type picker, create a tTTL resource (CMD-K).
2. In the Select New Type dialog, enter tTTL and click OK.
3. Set the title to "Details".
4. Set the resource Info (CMD-I).
5. Set the ID to 1301 and the Name to "Memo Details".
6. Close all the windows down to the type picker.

10.1.2.3 Creating Buttons for Details Form

This section has you create three buttons for the Details form:

- OK
- Cancel
- Delete

Like most dialogs, this one needs an OK button; follow these steps:

1. From the type picker, create a tBTN resource (CMD-K).
2. Set the Button ID to 1302.
3. Set the button's dimensions to 10, 52, 37, 12.
4. Set the Font to 0 and the Label to "OK".
5. Set resource Info (CMD-I).
6. Set the ID to 1302 and the Name to "OK".

With the tBTN resource picker still open, create another button resource for the Cancel button; follow these steps:

1. Create a tBTN resource (CMD-K).
2. Set the Button ID to 1303.
3. Set the dimensions to 61, 52, 37, 12.
4. Set the Font to 0 and Label to "Cancel".
5. Set resource Info; choose Resource>Get Resource Info (CMD-I).
6. Set the ID to 1303 and the Name to "Cancel".
7. Close the Info window, the tBTN "Cancel" editor window.

With the tBTN resource picker still open, create a button resource for the Delete button; follow these steps:

1. Create a tBTN resource (CMD-K).
2. Set the Button ID to 1304.
3. Set the dimensions to 111, 52, 37, 12.
4. Set the Font to 0 and Label to "Delete".
5. Set resource Info (CMD-I).

6. Set the ID to 1304 and the Name to “Delete”.
7. Close all the windows down to the type picker.

10.1.2.4 Creating a Label for the Secret Checkbox

Create a label (tLBL) resource for the secret checkbox; follow these steps:

1. From type picker, create a tLBL resource (CMD-K).
2. Set the Label ID to 1308
3. Set the label’s position to 13 and 33.
4. Set the Font to 1 and Text to “Private”.
5. Set resource Info (CMD-I).
6. Set the ID to 1308 and the Name to “Private”.
7. Close all the windows down to the type picker.

10.1.2.5 Creating a Help String

Create a string (tSTR) resource to contain the help text for the Details dialog; follow these steps:

1. Create a tSTR resource (CMD-K).
2. Set the String to “To hide private records, tap the Applications (arrow) icon and go to the Security application”.
3. Set resource Info (CMD-I).
4. Set the ID to 1305 and the Name to “Details:Help”.
5. Close all the windows down to the type picker.

10.1.2.6 Creating a Secret Checkbox

Create a checkbox (tCBX) resource for the secret checkbox; follow these steps:

1. Set the Check box ID to 1306.
2. Set dimensions to 55, 31, 14, 14.
3. Set Selected to False.
4. Set Group to 0.
5. Set Font to 0.
6. Leave the Label box empty.
7. Set resource Info (CMD-I).

8. Set the ID to 1306 and the Name to “Private”.
9. Close all the windows down to the type picker.

10.1.2.7 Putting Resources into the Details Form

Put the six new resources into the form; follow these steps:

1. Open the tFRM resource picker.
2. Open the Memo Pad Details form (ID 1300).
3. Set the Help Rsc ID to 1305 to set the connection to the help string resource.
4. Below the label “Number of Objs”, click the item labeled “1) *****” and insert a new field (CMD-K).
5. For the title resource; set the Obj ID to 1301 and the Obj Type to tTTL.
6. Click the item labeled “2) *****” and insert a new field (CMD-K).
7. For the OK button; set the Obj ID to 1302 and the Obj Type to tBTN.
8. Click the item labeled “3) *****” and insert a new field (CMD-K).
9. For the Cancel button; set the Obj ID to 1303 and the Obj Type to tBTN.
10. Click the item labeled “4) *****” and insert a new field (CMD-K).
11. For the Delete button; set ID to 1304 and the Obj Type to tBTN.
12. Click the item labeled “5) *****” and insert a new field (CMD-K).
13. For the private checkbox; set ID to 1306 and the Obj Type to tCBX.
14. Click the item labeled “6) *****” and insert a new field (CMD-K).
15. For the private label; set the Obj ID to 1308 and the Obj Type to tLBL.
16. Close all the windows down to the type picker.
17. Save the resource file.

10.1.3 Examining the Code

10.1.3.1 Adding #define Macros for New Resources

The programmer has added #define macros for the components of the details form to MemoPadRsc.h. In addition, the MemoPadRsc.c file was modified to include the MemoPadDetails.rsrc file in the list of application resource files. This phase adds event handlers for the Details button and for the Details dialog.

10.1.3.2 Adding Event Handlers

This phase adds event handlers for the Details button and for the Details dialog. To view the code, from CodeWarrior; open MemoPadRsc.h.

10.1.3.3 Adding an Event Handler for the Details Button

The EditFormHandleEvent routine has been modified to handle the new Details button. The programmer has added a test for the editDetailsButton. When this button is tapped, the details form is started by a call to FrmPopupForm. This routine generates a frmLoadEvent and a frmOpenEvent. It does not generate a frmCloseEvent before the other two events. This means the edit form is not erased before the details form is displayed. This has the effect of popping up the details form over the edit form.

10.1.3.4 Adding an Event Handler for the Details Dialog

The ApplicationHandleEvent routine has been modified to include a case for the detailsForm. FrmSetEventHandler is called to set up the application's new DetailsFormHandleEvent routine to receive events, while the details form is active.

The new routine, DetailsFormHandleEvent, processes events for the details form. It includes processing for each of the three buttons on the form and for the frmOpenEvent.

For the frmOpenEvent, the new DetailsInit routine is called to set up the form after which the form is displayed by a call to FrmDrawForm. The DetailsInit routine retrieves the setting for "private" for the current record, then sets the corresponding value for the private checkbox in the form.

A call to DmRecordInfo retrieves the attribute information for the current record. The attribute information is a collection of flags and other data packed into a byte. Each database record has an attribute byte that is stored and maintained by the system's database manager. One of the flags indicates that a record is private (secret). The system uses this flag to hide private records, if that preference has been set in the Security application.

The value of the private flag is stored in the record attribute. A call to the new routine GetObjectPtr gets a pointer to the private checkbox's data. The value of the checkbox control is set, based on the value of the secret flag in the record attributes. When the form is later displayed, the setting of the checkbox corresponds to the value in the record.

The DetailsFormHandleEvent routine contains a switch to handle the ctlSelectEvent for each button on the form. Tapping the OK button calls the new DetailsApply routine to apply changes made in the details form to the current record. Then, the details form is cleared, and the edit form displayed by a call to FrmReturnToForm. This routine erases and deletes the details form and then activates the edit form. FrmReturnToForm assumes that the edit form is already loaded into memory.

The DetailsApply routine retrieves the attribute of the current record and calls CtlGetValue to retrieve the setting of the secret checkbox on the details form. If the checkbox value is different from the setting in the record attribute, the user changed the checkbox on the details form. If so, the new value for the secret flag is set in the attribute byte. Also, the dirty flag is set in the attribute byte. This flag indicates to the database manager that the record contains data that has changed. Finally, the updated attribute byte is written to the database by calling DmSetRecordInfo.

In the DetailsFormHandleEvent routine, the case for the Cancel button has a simple task. Changes made in the details dialog are ignored. A call to FrmReturnToForm clears the details form and displays the edit form.

The case for the Delete button calls the new EditDeleteCurrentRecord routine which deletes the record, if the user confirms the request. If the record is deleted, a call is made to FrmCloseAllForms. This routine generates frmCloseEvents for all currently open forms

and sets no form to be active. In this case, that includes the details and the edit forms. Then, a call to FrmGotoForm starts the main form.

The EditDeleteCurrentRecord routine is actually the same code that had previously been contained within the case for the deletePage menu command in the EditDoMenuCommand routine. The new routine includes the return of a Boolean value to inform the caller if the record was deleted or not.

NOTE: The case for the deletePage command in EditDoMenu-Command has been revised to call the EditDeleteCurrentRecord routine.

10.2 Building and Exercising the Application

With resources and code in place for Memo Pad 10, build the project and exercise it in the Pilot Simulator; follow these steps:

1. With CodeWarrior running, check the Project menu to make sure that the debugger is disabled before running the application. It should show the item Enable Debugger.
2. Build the application (CMD-M).
3. Launch the application (CMD-R). The Pilot Simulator starts and displays the Pilot with your memo pad application in it.

With the application up and running, exercise the features added in this phase; follow these steps:

1. Tap the New button on the main form to cause the edit form to be displayed. Enter a few words into the text field.
2. Tap the Details button to display the details dialog.
3. Tap the checkbox labeled Private to check the box.
4. Tap the OK button to dismiss the details dialog and record the private setting in the database record.
5. Tap the Done button on the edit form to display the main form.

6. Tap the item in the list to display the edit form again with the previously entered text.
7. Tap the Details button to display the Details dialog and note that the checkbox is checked.
8. Tap the Delete button. Tap the Cancel button on the Delete Memo alert to return to the Details dialog.
9. Tap the Delete button again. Tap the OK button on the Delete Memo alert. The details and edit forms are erased and the main form is displayed with no records in the list.
10. Quit the application; choose the Simulator File>Quit item.

Adding Categories

Overview

In Phase 11, you add category assignment to the memos. The database contains slots for 16 categories some of which have predefined names. One of them is the default category “unfiled”. Each record is assigned to one of the categories. The Main form is enhanced to show all the records (as before) or show only those records in a single category.

A new dialog is added to allow the user to create and edit category names. The Edit and Details forms are enhanced to display and assign the category of the current record. See Figures 11-1 through 11-3.

To work this phase, follow these steps:

1. If MemoPadMain.rsrc, MemoPadInfo.rsrc, MemoPadEdit.rsrc and MemoPadDetails.rsrc do not already exist in Working Folder:Rsc from Phase 10, then copy them from the MemoPad 10:Rsc folder.
2. Copy MemoPad.c, MemoPadRsc.c, and MemoPadRsc.h from the MemoPad 11:Src folder to Working Folder:Src.
3. If MemoPad.µ does not exist in Working Folder from Phase 10, copy it from MemoPad 10.

Figure 11.1 Phase 11 Memo Pad Main form with Category selection.

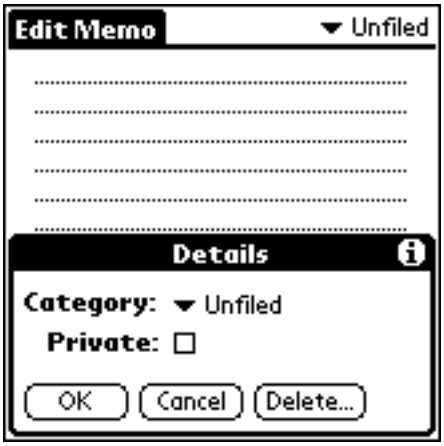

Figure 11.2 Phase 11 Memo Pad Details form with Category selection.

Figure 11.3 System-Provided Category Edit Form.

11.1 Adding Category UI to the Forms

Categories are a fundamental element of Palm OS databases. Under Palm OS, categories serve a similar function as files do on the desktop.

Record-based Palm OS applications generally offer category-based viewing of records in the list view, if any.

Applications usually allow the user to create and edit the categories, but it is also possible to pre-load a fixed set of categories. Applications also generally allow users to assign records to categories as they wish, but can, alternatively, do this internally based on program logic. This Memo Pad tutorial chooses the more basic approach.

11.1.1 Adding Category UI to the Main Form

This section shows you how to create a category trigger and popup selection list on the main form.

To begin, from ResEdit, open the resource file for the main form:

1. Open the Working Folder:Rsc folder.
2. Open the MemoPadMain.rsrc type picker.

11.1.1.1 Adding the Category Popup Trigger

Add the popup trigger to display the name of the current category and when tapped bring up the list of categories; follow these steps:

1. Create a new tPUT resource (CMD-K).
2. Set the Popup Trigger ID to 1007.
3. Specify position and size as follows: 160, 0, 0, and 13.
4. Set Left anchor to False.
5. Set Font to 0 (zero) and leave the Label empty.
6. Set the resource Info (CMD-I).
7. Set the ID to 1007 and the name to “Category”.
8. Close all the windows down to the type picker.

11.1.1.2 Adding the Category List

Add the list of categories; follow these steps:

1. Create a new tLST resource (CMD-K).
2. Set the List ID to 1008.
3. Specify position and size as follows: 86, 1, and 72.
4. Set Usable to False.
5. Set Font to 0 (zero).
6. Set Visible Items to 0 (zero).
7. Set the resource Info (CMD-I).
8. Set the ID to 1008 and the name to “Category List”.
9. Close all the windows down to the type picker.

11.1.1.3 Connecting the Popup Trigger to the List

Add the popup list which links the popup trigger to the list to be displayed; follow these steps:

1. Create a new tPUL resource (CMD-K).
2. Set the Control ID to 1007 and the List ID to 1008.
3. Set the resource Info (CMD-I).
4. Set the ID to 1006 and the name to “Categories”.
5. Close all the windows down to the type picker.

11.1.1.4 Attaching the Category UI to the Main Form

Add the Category UI to the main form; follow these steps:

1. Open the tFRM resource.
2. Open the Memo Pad Main form (ID 1000).
3. Click the item labeled "5) ******" and insert a new field (CMD-K).
4. For the popup list; set ID to 1006 and Type to tPUL.
5. Click the item labeled "6) ******" and insert a new field (CMD-K).
6. For the popup trigger; set ID to 1007 and Type to tPUT.
7. Click the item labeled "7) ******" and insert a new field (CMD-K).
8. For the list; set ID to 1008 and Type to tLST.
9. Close all the windows down to the type picker.
10. Save and close the resource file.

11.1.2 Adding Category UI to the Edit Form

This section shows you how to create a category trigger and popup selection list on the edit form.

To begin, from ResEdit, open the resource file for the edit form:

1. Open the Working Folder:Rsc folder.
2. Open the MemoPadEdit.rsrc type picker.

11.1.2.1 Creating the Category Popup Trigger

Add the popup trigger to display the name of the current category and when tapped bring up the list of categories; follow these steps:

1. Create a new tPUT resource (CMD-K).
2. Set the Popup Trigger ID to 1207.
3. Specify position and size as follows: 160, 0, 0, and 13.
4. Set Left anchor to False.
5. Set Font to 0 (zero) and leave the Label empty.
6. Set the resource Info (CMD-I).
7. Set the ID to 1207 and the name to "Category".
8. Close all the windows down to the type picker.

11.1.2.2 Creating the Category List

Add the list of categories; follow these steps:

1. Create a new tLST resource (CMD-K).
2. Set the List ID to 1208.
3. Specify position and size as follows: 86, 1, and 72.
4. Set Usable to False.
5. Set Font to 0 (zero).
6. Set Visible Items to 0 (zero).
7. Set the resource Info (CMD-I).
8. Set the ID to 1208 and the name to "Category List".
9. Close all the windows down to the type picker.

11.1.2.3 Connect the List to the Popup Trigger

The list object is associated with the popup trigger object by using a popup list object. Add the popup list which links the popup trigger to the list to be displayed; follow these steps:

1. Create a new tPUL resource (CMD-K).
2. Set the Control ID to 1207 and the List ID to 1208.
3. Set the resource Info (CMD-I).
4. Set the ID to 1206 and the name to "Categories".
5. Close all the windows down to the type picker.

11.1.2.4 Adding the Category UI to the Edit Form

Add the Category UI to the edit form; follow these steps:

1. Open the tFRM resource.
2. Open the Memo Pad Edit form (ID 1200).
3. Click the item labeled "5) ******" and insert a new field (CMD-K).
4. For the popup list; set ID to 1206 and Type to tPUL.
5. Click the item labeled "6) ******" and insert a new field (CMD-K).
6. For the popup trigger; set ID to 1207 and Type to tPUT.
7. Click the item labeled "7) ******" and insert a new field (CMD-K).

8. For the list; set ID to 1208 and Type to tLST.
9. Close all the windows down to the type picker.
10. Save and close the resource file.

11.1.3 Adding Category UI to the Details Dialog

This section shows you how to create a category trigger and popup selection list on the details form.

To begin, from ResEdit, open the resource file for the details form:

1. Open the Working Folder:Rsc folder.
2. Open the MemoPadDetails.rsrc type picker.

11.1.3.1 Creating the Category Popup Trigger

Add the popup trigger to display the name of the current category and when tapped bring up the list of categories; follow these steps:

1. Create a new tPUT resource (CMD-K).
2. Set the Popup Trigger ID to 1311.
3. Specify position and size as follows: 57, 18, 0, and 13.
4. Set Font to 0 (zero) and leave the Label empty.
5. Set the resource Info (CMD-I).
6. Set the ID to 1311 and the name to “Category”.
7. Close all the windows down to the type picker.

11.1.3.2 Creating the Category List

Add the list of categories; follow these steps:

1. Create a new tLST resource (CMD-K).
2. Set the List ID to 1312.
3. Specify position and size as follows:59, 19, and 72.
4. Set Usable to False.
5. Set Font to 0 (zero).
6. Set Visible Items to 0 (zero).
7. Set the resource Info (CMD-I).
8. Set the ID to 1312 and the name to “Category List”.
9. Close all the windows down to the type picker.

11.1.3.3 Connect the List to the Popup Trigger

Add the popup list which links the popup trigger to the list to be displayed; follow these steps:

1. Create a new tPUL resource (CMD-K).
2. Set the Control ID to 1311.
3. Set the List ID to 1312.
4. Set the resource Info (CMD-I).
5. Set the ID to 1310 and the name to "Categories".
6. Close all the windows down to the type picker.

11.1.3.4 Creating a Label for the Popup Trigger

Add a label for the category popup trigger; follow these steps:

1. Create a new tLBL resource (CMD-K).
2. Set the Label ID to 1309.
3. Specify position as follows: 4 and 18.
4. Set Font to 1.
5. Set Text to "Category:".
6. Set the resource Info (CMD-I).
7. Set the ID to 1309 and the name to "Category".
8. Close all the windows down to the type picker.

11.1.3.5 Attaching the Category UI to the Details Form

Add the Category UI to the details form; follow these steps:

1. Open the tFRM resource.
2. Open the Memo Pad Details form (ID 1300).
3. Click the item labeled "5) ******" and insert a new field (CMD-K).
4. For the label; set ID to 1309 and Type to tLBL.
5. Click the item labeled "6) ******" and insert a new field (CMD-K).
6. For the popup list; set ID to 1310 and Type to tPUL.
7. Click the item labeled "7) ******" and insert a new field (CMD-K).
8. For the popup trigger; set ID to 1311 and Type to tPUT.

9. Click the item labeled "8) ******" and insert a new field (CMD-K).
10. For the list; set ID to 1312 and Type to tLST.
11. Close all the windows down to the type picker.
12. Save and close the resource file.

11.1.4 Examining the Code

11.1.4.1 Adding #define Macros for New Resources

The programmer has added #define macros for the category UI components added to the main, edit and details forms to MemoPadRsc.h.

11.1.4.2 Adding Event Handlers

The system provides a high level user interface component to allow the user to display, select and edit the category names. This component frees the application from most of the work associated with managing category names. This phase adds event handling for the category in the Main, Edit and Details forms. To view the code, from CodeWarrior; open MemoPad.c.

11.1.4.3 Handling Categories for the Main Form

The StartApplication routine now calls MemoPadInitAppInfo when a new database is created.

The new MemoPadInitAppInfo routine installs default category names in the newly allocated application's information block (appInfo block). This is a standard technique for storage of category names and the system provides functions and user interface based on this design.

The appInfo block is an optional component of a Palm OS database. It is a good place for category names because it keeps the category names within the database. This means that changes to the category names are updated on the desktop during HotSync.

After the appInfo block is setup, the StartApplication routine calls the system routine CategoryGetName to get the current category name and category id from the appInfo block into global variables for the application's use.

The MainViewInit routine calls the system routines CategoryGetName and CategorySetTriggerLabel to set the category list trigger name to the name of CurrentCategory.

The functionality to load the list with the records to display has been extracted into its own routine, MainViewLoadRecords, so that it may be called from several locations.

In the MainViewLoadRecords routine a call to DmNumRecordsInCategory replaces the call to DMNumRecords. DmNumRecordsInCategory is passed to the category the user selected, and will ignore records in other categories.

Similarly, in MainViewListDrawItem the routine DmSeekRecordInCategory is now used because it will skip records in other categories. This creates a potential performance problem when drawing the list if the number of records is large. Finding the next record to draw means calling DmSeekRecordInCategory for a specific record index.

To find the requested record DmSeekRecordInCategory must start from record zero and advance forward n records while skipping records from other categories. This will slow down when the number of records is large.

If there are 1000 records and the list is showing the last 10 then it takes approximately 10 * 1000 record accesses to find the records to draw the list. The next phase shows an alternate implementation that avoids this problem by the use of tables.

When the user taps on the category trigger in the main form MainViewHandleEvent now calls MainViewSelectCategory. This new routine calls the system routine CategorySelect to handle the category selection and editing of category names.

If the user selects a different category MainViewSelectCategory updates the category UI, reloads the list with records from the new category and draws the new list contents.

11.1.4.4 Handling Categories for the Edit Form

The edit view has changes like the main view to set the category UI and handle its use. Changes to the selected category are made to the record and are kept in the new global variable RecordCategory.

EditViewUpdateDisplay is a new routine called by EditViewHandleEvent to process a frmUpdateEvent. This event is used to notify a form that it needs to be visually updated.

In this case the routine is needed to update the edit view when the user changes the category from the details dialog. The frmUpdateEvent is added to the event queue from the details dialog.

11.1.4.5 Handling Categories for the Details Form

The details dialog also includes new code to handle setting and changing the category UI. All changes are made to NewCategory instead of RecordCategory in case the Cancel button is pressed in the details dialog.

If the category does change, DetailsApply now changes the record and returns an update code. This code informs DetailsFormHandleEvent to send an update event to the edit form so it can properly update it's category UI.

The frmUpdateEvent is used to convey a change to a form that is not currently active. The event must arrive at the form's event handler routine after the form has become active and can handle the event. Consequently, the frmUpdateEvent is placed on the event queue after the frmOpenEvent.

The order of events is important because the UI for the edit form exists only after the frmOpenEvent is handled. EditViewHandleEvent needs to have a form to update when frmUpdateEvent arrives!

11.2 Building and Exercising the Application

With resources and code in place for Memo Pad 11, build the project and exercise it in the Pilot Simulator; follow these steps:

1. With CodeWarrior running, check the Project menu to make sure that the debugger is disabled before running the application. It should show the item Enable Debugger.
2. Build the application (CMD-M).
3. Launch the application (CMD-R). The Pilot Simulator starts and displays the Pilot with your memo pad application in it.

With the application up and running, exercise the features added in this phase; follow these steps:

1. Tap the New button on the main form to cause the edit form to be displayed. Enter a few words into the text field.
2. Tap the Details button to display the details dialog.
3. Tap the category name trigger to display the list of categories.
4. Tap the bottom entry in the list “Edit Categories...”.
5. Tap on the empty line under Personal and enter “Recreation”.
6. Tap on the Done button to dismiss the category edit dialog.
7. Tap the category name trigger and select Recreation from the list of categories.
8. Tap the OK button to dismiss the details dialog. Notice that the category name on the title of the Edit form is now Recreation.
9. Tap the Done button to dismiss the edit form and display the main form. Notice that since the category selection is “All” the new record is displayed in the list.
10. Tap the category name trigger and select Business. Notice that the list is now empty.
11. Tap the category name trigger and select Recreation. Notice that the record is now displayed.
12. Quit the application; choose the Simulator File>Quit item.

Using a Table to Display the Database

Overview

In Phase 12, you replace the list of records in the main form with a table. It is more efficient to use a table when displaying records with different categories because the record displayed is stored in the table, and the table items are loaded in one pass over the database. Tables have additional abilities like multiple columns and different types of items like popup lists or check boxes. This phase uses only a simple one column table to replace the list. See Figure 12-1.

To work this phase, follow these steps:

1. If MemoPadMain.rsrc, MemoPadInfo.rsrc, MemoPadEdit.rsrc and MemoPadDetails.rsrc do not already exist in Working Folder:Rsc from Phase 11, then copy them from the MemoPad 11:Rsc folder.
2. Copy MemoPad.c, MemoPadRsc.c, and MemoPadRsc.h from the MemoPad 12:Src folder to Working Folder:Src.
3. If MemoPad.µ does not exist in Working Folder from Phase 11, copy it from MemoPad 11.

Figure 12.1 Phase 12 Memo Pad Main form with table based record display.

12.1 Adding Table UI to the Main Form

Tables offer a more sophisticated mechanism for viewing lists of data. You will generally want to use a table instead of a list when displaying large amounts of data, or when used for viewing editable or highly formatted data. Tables also give you better control over scrolling than lists.

12.1.1 Removing the Record List UI from the Main Form

This section shows you how to remove the record display list from the main form.

To begin, from ResEdit, open the resource file for the main form:

1. Open the Working Folder:Rsc folder.
2. Open the MemoPadMain.rsrc type picker.

Remove the record list from the form and the resource file; follow these steps:

1. Open the tFRM resource.
2. Open the Memo Pad Main form (ID 1000).

3. Click the item labeled "4) ******"
4. Cut the item (CMD-X).
5. Close all the windows down to the type picker.
6. Open the tLST resource.
7. Click the Memo list (ID 1005)
8. Cut the item (CMD-X).
9. Close all the windows down to the type picker.

12.1.2 Adding the Table UI to the Main Form

Add the table to display the records; follow these steps:

1. Create a new tTBL resource (CMD-K).
2. Set the Table ID to 1005.
3. Specify position and size as follows: 0, 18, 160 and 121.
4. Set Editable to False.
5. Set Rows to 11
6. Click the item labeled "1) ******" and insert a new field (CMD-K).
7. Set Column Width to 160.
8. Set the resource Info (CMD-I).
9. Set the ID to 1005 and the name to "Memo Table".
10. Close all the windows down to the type picker.

Add the table to the main form; follow these steps:

1. Open the tFRM resource.
2. Open the Memo Pad Main form (ID 1000)
3. Click the item labeled "4) ******" and insert a new field (CMD-K).
4. For the table; set ID to 1005 and Type to tTBL.
5. Close all the windows down to the type picker.
6. Save and close the resource file.

12.1.3 Examining the Code

12.1.3.1 Adding #define Macros for New Resources

The programmer has replaced the #define macro for mainList with one for the new table in MemoPadRsc.h.

12.1.3.2 Adding Event Handlers

This phase adds event handling for the table in the Main form. To view the code, from CodeWarrior; open MemoPad.c.

12.1.3.3 Handling the Table in the Main Form

The MainViewLoadRecords routine has been revised to work with the table object. The first task is to position the table over the database by setting the value of TopVisibleRecord to the index of one of the database records.

The intent is to ensure that the CurrentRecord will be visible and that all rows displayed in the table have a record (to make the most of the screen space). Once the top record is set, the table is loaded with records by the new routine MainViewLoadTable.

In MainViewLoadTable, for each row of the table, a database record in the CurrentCategory is found by a call to DmQueryNextInCategory.

The record's index is assigned to the table row's value for column zero. This value will be used later when the table row is drawn. Rows without records (because there are too few records in the current category to fill the table) are marked not usable.

Once the table is loaded with appropriate record indexes, MainViewLoadRecords sets the application routine that will be used to draw each row of the table. The system's table handling routine will call MainViewDrawRecord to draw each one of its visible table items. This new routine replaces and is similar to MainViewListDrawItem but it works with a table instead of a list.

Lastly, in MainViewLoadRecords the first, and only, column of this table is set usable so that it can be drawn.

In MainViewHandleEvent the case for lstSelectEvent has been replaced by a case for tblSelectEvent. The record index is retrieved

from the table's value for column zero of the current row and assigned to CurrentRecord.

12.2 Building and Exercising the Application

With resources and code in place for Memo Pad 12, build the project and exercise it in the Pilot Simulator; follow these steps:

1. With CodeWarrior running, check the Project menu to make sure that the debugger is disabled before running the application. It should show the item Enable Debugger.
2. Build the application (CMD-M).
3. Launch the application (CMD-R). The Pilot Simulator starts and displays the Pilot with your memo pad application in it.

With the application up and running, exercise the features added in this phase; follow these steps:

1. Tap the New button on the main form to cause the edit form to be displayed. Enter a few words into the text field.
2. Return to the main form; tap the Done button. The main form now displays a table with a single item.
3. Tap the New button to create another new record edit form. Enter a few words into the text field.
4. Return to the main form; tap the Done button. The main form now displays a table with two items.
5. Tap either item and, in the edit form, delete all the text of the record.
6. Return to the main form; tap the Done button. The main form again displays a table with a single item. The Palm OS system has removed the item from the table and from the database.
7. Quit the application; choose the Simulator File>Quit item.

Adding Scrolling to the Main and Edit Forms

Overview

In Phase 13, you add user controllable scrolling of the main and edit forms. For the main form this means repositioning and reloading the table object as it scrolls up and down. For the edit form the field object can be instructed to scroll. Both forms have scroll buttons added to the UI and event handlers to handle the buttons.

To work this phase, follow these steps:

1. If MemoPadMain.rsrc, MemoPadInfo.rsrc, MemoPadEdit.rsrc and MemoPadDetails.rsrc do not already exist in Working Folder:Rsc from Phase 12, then copy them from the MemoPad 12:Rsc folder.
2. Copy MemoPad.c, MemoPadRsc.c, and MemoPadRsc.h from the MemoPad 13:Src folder to Working Folder:Src.
3. If MemoPad.µ does not exist in Working Folder from Phase 12, copy it from MemoPad 12.

Figure 13.1 Phase 13 Memo Pad Main form with scrolling arrows.

Figure 13.2 Phase 13 Memo Pad Edit form with scrolling arrows.

13.1 Adding Scrolling to the Forms

You will want to add scrolling capability to almost any display of data whether in a list, table, or field. Scrolling can involve on-screen arrow buttons (repeating or non-repeating), the physical scroll buttons, or both.

Scrolling can be done by line, record, or page. Horizontal scrolling is not natively supported by Palm OS objects, although it can be used with custom-drawn UI, such as the Pilot Date Book's Week View.

13.1.1 Adding Scroll Arrows to the Main Form

This section shows you how to add repeating scroll arrows to the main form.

To begin, from ResEdit, open the resource file for the main form:

1. Open the Working Folder:Rsc folder.
2. Open the MemoPadMain.rsrc type picker.

13.1.1.1 Creating the Scroll Arrows

Add the up and down scroll arrows; follow these steps:

1. Create a new tREP resource (CMD-K).

2. Set the Button ID to 1009.
3. Specify position and size as follows: 147, 144, 13, and 8.
4. Set Frame to False.
5. Set Font to 5, for the symbol font, and leave the Label empty.
6. Set the resource Info (CMD-I).
7. Set the ID to 1009 and the name to "Scroll Up".
8. Close all the windows down to the type picker.
9. Create a second new tREP resource (CMD-K).
10. Set the Button ID to 1010.
11. Specify position and size as follows: 147, 152, 13, and 8.
12. Set Frame to False.
13. Set Font to 5 and leave the Label empty.
14. Set the resource Info (CMD-I).
15. Set the ID to 1010 and the name to "Scroll Down".
16. Close all the windows down to the type picker.

13.1.1.2 Attaching the Scroll Arrows UI to the Main Form

Add the scroll arrows UI to the main form; follow these steps:

1. Open the tFRM resource.
2. Open the Memo Pad Main form (ID 1000).
3. Click the item labeled "8) ******" and insert a new field (CMD-K).
4. For the up arrow; set ID to 1009 and Type to tREP.
5. Click the item labeled "9) ******" and insert a new field (CMD-K).
6. For the popup trigger; set ID to 1010 and Type to tREP.
7. Close all the windows down to the type picker.
8. Save and close the resource file.

13.1.2 Adding Scroll Arrows to the Edit Form

This section shows you how to add repeating scroll arrows to the main form.

To begin, from ResEdit, open the resource file for the main form:

1. Open the Working Folder:Rsc folder.

2. Open the MemoPadEdit.rsrc type picker.

13.1.2.1 Creating the Up and Down Scroll Arrows

Add the up and down scroll arrows; follow these steps:

1. Create a new tREP resource (CMD-K).
2. Set the Button ID to 1209.
3. Specify position and size as follows: 147, 144, 13, and 8.
4. Set Frame to False.
5. Set Font to 5 and leave the Label empty.
6. Set the resource Info (CMD-I).
7. Set the ID to 1209 and the name to “Scroll Up”.
8. Close all the windows down to the type picker.
9. Create a second new tREP resource (CMD-K).
10. Set the Button ID to 1210.
11. Specify position and size as follows: 147, 152, 13, and 8.
12. Set Frame to False.
13. Set Font to 5 and leave the Label empty.
14. Set the resource Info (CMD-I).
15. Set the ID to 1210 and the name to “Scroll Down”.
16. Close all the windows down to the type picker.

13.1.2.2 Attaching the Scroll Arrows UI to the Edit Form

Add the scroll arrows UI to the edit form; follow these steps:

1. Open the tFRM resource.
2. Open the Memo Pad Edit form (ID 1200).
3. Click the item labeled “9) ******” and insert a new field (CMD-K).
4. For the up arrow; set ID to 1209 and Type to tREP.
5. Click the item labeled “10) ******” and insert a new field (CMD-K).
6. For the popup trigger; set ID to 1210 and Type to tREP.
7. Close all the windows down to the type picker.
8. Save and close the resource file.

13.1.3 Examining the Code

13.1.3.1 Adding #define Macros for New Resources

The programmer has added #define macros for the scroll arrow UI components added to the main and edit forms to MemoPadRsc.h.

13.1.3.2 Handling Scrolling Arrows for the Main Form

Two new routines, MainViewScroll and MainViewUpdateScrollers, have been added to scroll the records in the main form's table and to control which scroll arrows are on or off.

To scroll up or down, MainViewScroll simply moves n records from the TopVisibleRecord. If there is a problem scrolling, the code tries to set TopVisibleRecord so that the table is filled (no blank rows).

MainViewUpdateScrollers turns the scroll arrows on and off. It is called whenever the table is loaded or repositioned.

The MainViewHandleEvent routine handles the scroll buttons and physical scroll buttons with a new case for ctlRepeatEvent. The UI scroll buttons are repeating controls which are a variation of the normal control object. The difference is that as long as the user presses the button a ctlRepeatEvent is queued approximately once a second. The application processes the ctlRepeatEvent by scrolling the table up or down by a call to MainViewScroll.

Note that the handled variable is not set to true in this case. The ctlRepeatEvent must be passed on to the system so that the repeating control handler can generate the next instance of the event.

13.1.3.3 Handling Scroll Arrows for the Edit Form

For the edit form, the new routine EditViewScroll lets FldScrollField do the work of scrolling the field object. It then calls EditViewUpdateScrollers to turn the edit form's scroll arrows on or off.

Calls to EditViewUpdateScrollers have to be added where ever the field is changed. That means the menu cut and paste commands, the goto commands, and when the record is loaded.

These updates are necessary because every change to the text of the field, including word wrapping, may cause a line to be added or removed from the field. This should be reflected by the state of the scroll arrows.

The EditViewHandleEvent routine is changed, in similar fashion to the event handler, for the main form to handle the repeating scroll buttons and the physical scroll buttons. In addition, because this view contains the scrolling text field, entry of new characters into the field must be monitored as well.

New characters from Graffiti are seen by the application as keyDownEvents that contain the new character. Every new character can potentially change the height of the field. The field handler is forced to process the character immediately by calling FrmHandleEvent. Then EditViewUpdateScrollers is called to update the scroll arrows, if necessary.

13.2 Building and Exercising the Application

With resources and code in place for Memo Pad 13, build the project and exercise it in the Pilot Simulator; follow these steps:

1. With CodeWarrior running, check the Project menu to make sure that the debugger is disabled before running the application. It should show the item Enable Debugger.
2. Build the application (CMD-M).
3. Launch the application (CMD-R). The Pilot Simulator starts and displays the Pilot with your memo pad application in it.

With the application up and running, exercise the features added in this phase; follow these steps:

1. Tap the New button on the main form to cause the edit form to be displayed. Enter a few words into the text field.
2. Tap the Done button to dismiss the edit form and display the main form.
3. Create at least 11 more records. Hint: use the Copy, Paste and New Page menu commands available in the edit form to do this more easily.
4. Tap the Done button to dismiss the edit form and display the main form. Note that there are now scroll arrows on the bottom right corner of the form.
5. Tap on the scroll arrows to scroll the records displayed in the table. Note that tapping the up and down keys has similar effect.

6. Tap on one of the records to display the edit view.
7. Enter at least 20 lines of text into the text field. Hint: use the Copy and Paste menu commands.
8. When there are more lines than can be displayed the scroll arrows will be displayed in the bottom right corner of the form.
9. Tap on the scroll arrows to scroll the text field. Note that tapping the up and down keys moves the text field a full screen at a time.
10. Quit the application; choose the Simulator File>Quit item.

Adding System Find Support

Overview

In Phase 14, you will add the system find capability. The system provides high level user interface components to allow the user to enter text to search for and display the results of the search. These components reduce the work that the application has to perform for the search function. See Figures 14-1 and 14-2.

To work this phase, follow these steps:

1. If MemoPadMain.rsrc, MemoPadInfo.rsrc, MemoPadEdit.rsrc and MemoPadDetails.rsrc do not already exist in Working Folder:Rsc from Phase 13, then copy them from the MemoPad 13:Rsc folder.
2. Copy MemoPad.c, MemoPadRsc.c, and MemoPadRsc.h from the MemoPad 14:Src folder to Working Folder:Src.
3. If MemoPad.μ does not exist in Working Folder from Phase 10, copy it from MemoPad 13.

Figure 14.1 Phase 14 System Find dialog.

Figure 14.2 Phase 14 System Find dialog.

14.1 Adding System Find

Any application that contains text data should support the system-wide find functionality of Palm OS. Participating in this feature involves three main responsibilities: responding to application launch codes, searching your own database, and navigating to display specific data. The system provides all of the user interface and logic for the find feature.

14.1.1 Adding a Find Header String

This section shows you how to add a string to the main form.

To begin, from ResEdit, open the resource file for the main form:

1. Open the Working Folder:Rsc folder.
2. Open the MemoPadMain.rsrc type picker.

Add a string containing the header for the records that will be displayed in the find results dialog; follow these steps:

1. Create a new tSTR resource (CMD-K).
2. Set The String to "Memos".

3. Set the resource Info (CMD-I).
4. Set the ID to 100 and the name to “Find Header Title”.
5. Close all the windows down to the type picker.
6. Save and close the resource file.

14.1.2 Examining the Code

14.1.2.1 Adding #define Macros for New Resources

The programmer has added #define macros for the Find header string to MemoPadRsc.h.

14.1.2.2 Adding Event Handlers

This phase adds event handling and other processing for the system Find functions. To view the code, from CodeWarrior; open MemoPad.c.

The system issues two commands to each application to perform a find. The first command is a search command. Upon a search command the application needs to search its database for records that contain the desired text string.

The second command is the goto command. A goto command instructs the application to display the record found in the application’s record edit view. There is a standard mechanism for the application to receive these and other commands from the system.

14.1.2.3 Handling the Search Command

The search and goto commands arrive at the application as calls to PilotMain, not as events! In essence, the application is launched by the system to process these and other special commands. These system launch commands and associated information are provided as arguments to PilotMain.

PilotMain has been revised to check the cmd argument for a system launch command. Normally, the application is launched with the normal Launch command after another application has stopped.

For other system launch commands the application may or may not already be running when PilotMain is called. The application code that handles special commands **must not** use global variables set up

by the application in StartApplication or elsewhere, because the StartApplication routine has not been executed.

To process the search command the application must search through all its database records and report any matches of the search string. The search should not include records that are currently hidden.

The search results could include many records and be difficult to display and read if a large list of matches were displayed. Instead, the system incorporates the ability to halt a search when the search results display is full, as well as resume the search at the same point later. All of this is handled by cooperation between the system and the application.

A Search routine has been added to search for a text string in the application's database. Since this routine can't rely on the application to be already running it must first locate and open its own database.

It then calls FindDrawHeader to provide a string to the system Find handler, that will be displayed as a heading for records from this application in the Find Results dialog.

Starting with the record index provided in the search launch command each record is obtained from the database and searched for the desired string.

Each part or field of the database record that needs searching should be passed to FindStrInStr. In this application there is only a single field containing the text of the record. If a match is found FindSaveMatch is called to inform the system Find handler.

It is possible for an application to pass to FindSaveMatch values in fieldNum or appCustom to indicate which part of the record contained the match. This information will be passed back to the application as arguments of the goto command. This can be useful for the application to position the display to a specific location within the record when it receives the goto command.

MainViewDrawRecordInBounds is a new routine that has been extracted from the MainViewDrawRecord routine. The drawing bounds are those of the Find Results dialog.

Each application should have its own version of DrawRecordInBounds, that displays the information from the record that is most useful to the user to identify the record in the Find Results dialog.

Note how each of Palm OS device's built in applications displays search results in this dialog.

The Search routine loops to scan each of the database records until there are no more records or until FindSaveMatch indicates that the Find Results dialog is full. Before it returns the Search routine must close its database.

14.1.2.4 Handling the Goto Command

A GoToItem routine has been added to handle the user tapping the Go to button in the Find Results dialog. The record specified in the goto command parameters must be displayed.

Note that in PilotMain when the goto command arrives a check is made to discover if the application is already running. The user could have initiated the search from this or any other application.

If this application is not already running, the StartApplication routine is called to get the application in a running state before calling GoToItem and starting the event loop.

In the GoToItem routine if the application is already running all the open forms are closed before proceeding. The edit form is started by creating and queuing a frmLoadEvent.

Then a frmGotoEvent is queued to display the matching record in the edit form. This event is loaded with the index of the record, the location where the text was found and information about the search string, all from the arguments of the goto command.

Other applications will load their own version of a record view form and they may pass more information to display the match (like the fieldNum or appCustom values).

For each form which can receive frmGotoEvent a handler must be added. A new case for frmGotoEvent is added to EditViewHandleEvent. The matching record is obtained from the database and the field is scrolled to ensure that the matching text is visible. The text is highlighted by setting the field selection.

In addition to the search and goto commands, a third command, SaveData is now handled by PilotMain to save the application data. This is important for certain cases where the database needs to be saved before it can be updated.

14.2 Building and Exercising the Application

With resources and code in place for Memo Pad 14, build the project and exercise it in the Pilot Simulator; follow these steps:

1. With CodeWarrior running, check the Project menu to make sure that the debugger is disabled before running the application. It should show the item Enable Debugger.
2. Build the application (CMD-M).
3. Launch the application (CMD-R). The Pilot Simulator starts and displays the Pilot with your memo pad application in it.

With the application up and running, exercise the features added in this phase; follow these steps:

1. Tap the New button on the main form to cause the edit form to be displayed. Enter a few words into the text field including the word “Pilot”.
2. Tap the Done button to dismiss the edit form and display the main form.
3. Create several more records, including some that include the word “Pilot”.
4. Tap the Find icon on the Simulator’s display.
5. Enter the word “Pilot” and tap the OK button. The Find Results dialog is displayed with a list that includes several memos.
6. Select one and tap the Go to button. The Edit form is displayed containing the record and find string is selected.
7. Quit the application; choose the Emulator File>Quit item.

Saving Program Settings Between Executions

Overview

In Phase 15, you add the ability for the application to save and restore program settings. The MemoPad will save the current record and category displayed and return the user to them when the application is launched the next time. This information is saved in the system's preferences database.

To work this phase, follow these steps:

1. If MemoPadMain.rsrc, MemoPadInfo.rsrc, MemoPadEdit.rsrc and MemoPadDetails.rsrc do not already exist in Working Folder:Rsc from Phase 14, then copy them from the MemoPad 14:Rsc folder.
2. Copy MemoPad.c, MemoPadRsc.c, and MemoPadRsc.h from the MemoPad 15:Src folder to Working Folder:Src.
3. If MemoPad.μ does not exist in Working Folder from Phase 14, copy it from MemoPad 14.

No new resources are necessary for this phase.

15.1 Saving the Program's State

Many applications will find it desirable to save varying amounts of state information from one execution to the next. This can range from saving current pages or viewing options to saving bit images of game animations.

Saving and reloading this information will significantly enhance the usability of your user interface and contribute to the integrated, seamless appearance of the device as a whole.

15.1.1 Examining the Code

15.1.1.1 Adding Event Handlers

The system includes a Preferences database that is used by all applications to store their options and preferences settings that are not dependent on the actual data records, such as what is to be displayed at startup.

To view the code, from CodeWarrior; open MemoPad.c.

The information that we wish to save is currently stored in some of the application's global variables.

First, physically separate the global variables we wish to save from the others in the file. This clarifies which variables are being saved and which aren't being saved and makes it easier to know where to put a new variable when you later add variables. A new structure, MemoPadPreferenceType, is defined to contain only the saved variables.

The StopApplication routine is enhanced to save application variables to the Preferences database just before the application stops. All applications store their persistent settings as a single, application specific resource within that database.

There is system support for this which takes a block of memory and writes it to the database. Therefore, we copy all the variables we want preserved to the MemoPadPreferencesType structure and then pass the structure to PrefSetAppPreferences to save it in the database.

StartApplication needs to read the saved variables when the application starts. PrefGetAppPreferences is called to read them from the Preferences database into the MemoPadPreferenceType structure. A verification of the saved record number is done to ensure that it is still available in the database. If not, the default values for the saved variables are used.

15.2 Building and Exercising the Application

With resources and code in place for Memo Pad 15, build the project and exercise it in the Pilot Simulator; follow these steps:

1. With CodeWarrior running, check the Project menu to make sure that the debugger is disabled before running the application. It should show the item Enable Debugger.
2. Build the application (CMD-M).
3. Launch the application (CMD-R). The Pilot Simulator starts and displays the Pilot with your memo pad application in it.

Testing the saving and restoring of application preferences is a little tricky because of when the settings are written to the Preferences database.

The application's database can be saved by the Pilot Simulator menu command Save Card. This is not sufficient for Preferences because the settings are written out by the application's StopApplication routine only when the Emulator itself closes.

To save data after the StopApplication routine use the Simulator's File:Save Before Quitting menu command. This command writes the memory card image after PilotMain returns and before the Simulator closes.

With the application up and running, exercise the features added in this phase; follow these steps:

1. Tap the New button on the main form to cause the edit form to be displayed. Enter a few words into the text field.
2. Tap the category name trigger to display the list of categories.
3. Tap the "Business" entry to set the category for the entry.
4. Tap the Done button to dismiss the edit form and display the main form.
5. Tap the category name trigger and select Business.
6. Tap the Simulator's File:Save Before Quitting menu command.
7. Quit the Simulator.
8. Restart the Simulator for MemoPad. Notice that the category setting in the main form has been restored to Business.

9. Create several more records, at least 12. Scroll to the bottom of the list.
10. Tap the Simulator's File:Save Before Quitting menu command.
11. Quit the Simulator.
12. Restart the Simulator for MemoPad. Notice that the scroll position of the main form has been restored to the bottom of the list.
13. Quit the application; choose the Simulator File>Quit item.

Flashy Features

Overview

In Phase 16, you add a few extra features to spice up the application. The edit view title is improved to include “n of n records” to show the position of the current record in the list of records. The edit view also includes buttons to change the display font and now automatically sets the Graffiti shift state when a new memo is created. See Figure 16-1.

To work this phase, follow these steps:

1. If MemoPadMain.rsrc, MemoPadInfo.rsrc, MemoPadEdit.rsrc and MemoPadDetails.rsrc do not already exist in Working Folder:Rsc from Phase 15, then copy them from the MemoPad 15:Rsc folder.
2. Copy MemoPad.c, MemoPadRsc.c, and MemoPadRsc.h from the MemoPad 16:Src folder to Working Folder:Src.
3. If MemoPad.μ does not exist in Working Folder from Phase 15, copy it from MemoPad 15.

Figure 16.1 Phase 16 Memo Pad Edit form with improved title and font selection.

16.1 Adding UI Elements to the Edit Form

In this section you will use ResEdit to create all of the UI resources for the improved title and font selection buttons of the edit form.

16.1.1 Adding Resources for the Edit View Title

This section shows you how to create a dynamic title for the edit form.

To begin, from ResEdit, open the resource file for the main form:

1. Open the Working Folder:Rsc folder.
2. Open the resource file for the edit form.

Add the string that will serve as a template to be filled in for the edit form title; follow these steps:

1. Create a new tSTR resource (CMD-K).
2. Set The String to "Memo # of #" and leave Data empty.
3. Set the resource Info (CMD-I).
4. Set the ID to 1000 and the name to "Edit View Title Template String".
5. Close all the windows down to the type picker.
6. Save and close the resource file.

NOTE: It is common for a developer to consolidate all the string resources (tSTR), that are used in an application in a single file, to make them easier to find.

16.1.2 Modifying the Text Field Resource

This section shows you how to resize the edit form's field resource, to accommodate the larger font size that will be introduced later in this phase.

To begin, from ResEdit, open the resource file for the edit form:

1. Open the Working Folder:Rsc folder.

2. Open the MemoPadEdit.rsrc type picker.

16.1.2.1 Shrinking the Field Resource

To reduce the vertical size of the field resource such that it will accommodate the large font, follow these steps:

1. Open the tFLD resource picker.
2. Open the Edit Field edit window.
3. Change the Height field from 130 to 127.
4. Close all windows down to the type picker.
5. Save and close the resource file.

16.1.3 Adding Resources for the Edit View Font Selection

This section shows you how to create font controls on the edit form.

To begin, from ResEdit, open the resource file for the edit form:

1. Open the Working Folder:Rsc folder.
2. Open the MemoPadEdit.rsrc type picker.

16.1.3.1 Creating the Font Push Buttons

Add two push buttons to select the font; follow these steps:

1. Create a new tPBN resource (CMD-K).
2. Set the Push Button ID to 1211.
3. Specify position and size as follows: 95, 147, 14, and 12.
4. Set Group to 1.
5. Set Font to 0 (zero) and the Label to “A”.
6. Set the resource Info (CMD-I).
7. Set the ID to 1211 and the name to “Small Font”.
8. Close all the windows down to the type picker.
9. Create a new tPBN resource (CMD-K).
10. Set the Push Button ID to 1212.
11. Specify position and size as follows: 110, 147, 14, and 12.
12. Set Group to 1.
13. Set Font to 2 and the Label to “A”.

14.Set the resource Info (CMD-I).

15.Set the ID to 1212 and the name to “Large Font”.

16.Close all the windows down to the type picker.

16.1.3.2 Adding the Font Push Buttons to the Edit Form

Add the font selection UI to the edit form; follow these steps:

1. Open the tFRM resource.
2. Open the Memo Pad Edit form (ID 1200)
3. Click the item labeled “11) ******” and insert a new field (CMD-K).
4. For the small-font push button; set ID to 1211 and Type to tPBN.
5. Click the item labeled “6) ******” and insert a new field (CMD-K).
6. For the large-font push button; set ID to 1212 and Type to tPBN.
7. Close all the windows down to the type picker.
8. Save and close the resource file.

16.1.4 Examining the Code

16.1.4.1 Adding #define Macros for New Resources

The programmer has added #define macros for the large and small font buttons and for the title template string to MemoPadRsc.h.

16.1.4.2 Adding Event Handlers

To view the code, from CodeWarrior; open MemoPad.c.

16.1.4.3 Handling the Dynamic Title for the Edit Form

The EditViewSetTitle routine is added to format and set the title for the edit form. This will set the title to indicate the position of the record within the current category (which may be All). We use a template string to do this.

The template contains two ‘#’ chars. The first one is replaced by the record's position and the second by the number of records in the cat-

egory. Using a template string helps when the application needs to be localized for another language.

Someone can now use ResEdit to change the text of the title completely for a new language but still have the numbers show up positioned correctly, without changing any code! Even more thorough would be to use two different replacement chars, e.g. '#' and '%' to allow reordering of the arguments within the template string.

Memory space for the new title is allocated on the dynamic heap and must be freed when the form is closed (frmCloseEvent in EditViewHandleEvent). This is preferable to leaving extra padding in the template title string because that complicates localization.

EditViewSetTitle now determines the numbers for the edit view title. The position is available with DmPositionInCategory. The count is determined by calling DmNumRecordsInCategory. The direct and simple approach is to get the count whenever the edit view is entered. However, if there are a large number of records in many categories this could be lengthy.

A better alternative is to call DmNumRecordsInCategory whenever the category selection changes, and then modify the number as we add and remove records. This second approach is used in the application.

We keep count in the new global variable MemosInCategory. A new define value MemosInCategoryUnknown is used to indicate that the count in MemosInCategory is not valid and should be obtained by EditViewSetTitle when the title needs to be displayed. When MemosInCategory contains a valid count it is modified directly as records are added and removed.

16.1.4.4 Handling the Font Selection for the Edit Form

To handle the font buttons a new test is added to EditViewHandleEvent for a ctlSelectEvent from either of the font push buttons.

The new routine EditViewChangeFont is called with the control ID of the push button. EditViewChangeFont sets the field's font by calling FldSetFont.

Changing the font can cause the number of lines in the field to change because of the different sizes of the fonts. So EditViewUpdateScrollers is called to update the scrollers for the field.

EditViewChangeFont remembers the font in the new global variable EditViewFont. This allows the application to remember the font setting after the edit view is closed and restore it the next time the edit view is displayed.

EditViewHandleEvent has been changed for frmOpenEvent and frmGotoEvent to set the state of the font selection push buttons to match the value in EditViewFont. The actual font for the field is set in EditViewRetrieveData after the text field is loaded with the record data.

16.1.4.5 Handling the Automatic Set of the Graffiti Shift State When a New Memo is Created

Setting the Graffiti shift state is sometimes convenient for the user. We set it to be in temporary up shift state each time a new memo is created by adding a call to GrfSetState in the CreateRecord routine.

16.2 Building and Exercising the Application

With resources and code in place for Memo Pad 16, build the project and exercise it in the Pilot Simulator; follow these steps:

1. With CodeWarrior running, check the Project menu to make sure that the debugger is disabled before running the application. It should show the item Enable Debugger.
2. Build the application (CMD-M).
3. Launch the application (CMD-R). The Pilot Simulator starts and displays the Pilot with your memo pad application in it.

With the application up and running, exercise the features added in this phase; follow these steps:

1. Tap the New button on the main form to cause the edit form to be displayed.
   Note that the title of the view now includes the number and count of memos. Also note, that the Graffiti shift indicator shows Graffiti to be in temporary up shift state. Enter a few words into the text field.
2. Tap the font selection push button for the larger font. Note the change in display of the field's text.

3. Tap the Done button to dismiss the edit form and display the main form.
4. Tap a record in the list to display it in the edit form. Notice that the field is displayed with the larger font and that the corresponding font selection button is selected.
5. Quit the application; choose the Emulator File>Quit item.

Working With the Desktop

Overview

In Phase 17, you extend the application to work well with the desktop computer. Most of the work of synchronization is handled automatically by the system and requires nothing extra of an application.

However, an application's database can be modified by the synchronization process, and the app must be able to handle such changes the next time it is launched. In addition, applications should support the deletion of records on the desktop once they are removed from the Palm OS device. There are also application launch commands to support when working with the desktop.

To work this phase, follow these steps:

1. If MemoPadMain.rsrc, MemoPadInfo.rsrc, MemoPadEdit.rsrc and MemoPadDetails.rsrc do not already exist in Working Folder:Rsc from Phase 16, then copy them from the MemoPad 16:Rsc folder.
2. Copy MemoPad.c, MemoPadRsc.c, and MemoPadRsc.h from the MemoPad 17:Src folder to Working Folder:Src.
3. If MemoPad.μ does not exist in Working Folder from Phase 16, copy it from MemoPad 16.

No new resources are necessary for this phase.

17.1 Integrating with HotSync

All of the changes required for this phase are in the code; no new resources. Look at the code changes carefully. The tight integration of the desktop HotSync Manager and device-resident Palm OS applications is a key feature of Palm OS and should be taken advantage of as much as possible.

Note that the application itself is completely isolated from the synchronization logic and the communications details. The extent of the application's interaction with the desktop is the handling of application launch codes (syncNotify and initDatabase), and the maintenance of the record attribute flags in the standard database records; if you handle both of these areas, the system and the desktop software (your conduit) handle the rest.

Examining the Code 17.1.1

17.1.1 Adding Event Handlers

To view the code, from CodeWarrior; open MemoPad.c.

17.1.1.1 Handling Delete for the Desktop

As an alternative to simply deleting records locally, Palm OS applications ensure that records deleted from the Palm OS device will be deleted on the desktop as well. This is an important part of synchronization with the desktop.

Internally (within the Palm OS databases), this is accomplished by deleting only the body of a record and leaving a stub in the database index. This stub contains the record's unique ID and some attribute flags. Desktop conduits use this information to delete, and possibly archive, the record from the desktop files. The stub will finally be removed as part of the conduit's cleanup procedure.

EditViewSaveData has been revised to use this new method by replacing DmRemoveRecord with DmDeleteRecord. This routine sets a deleted flag in the record attributes and frees the rest of the record's data.

These 'deleted' stub records will be ignored by routines such as DmSeekRecordInCategory and DmQueryRecord.

As an alternative to simply deleting records, Palm OS applications typically allow records to be deleted and archived on the desktop for later reference. This reduces the data on the Palm OS device, to those records currently used, while removed records are still accessible on the desktop when they're occasionally needed. (Ideally, applications should provide both functions and let the user determine

whether or not to archive a deleted record, as Palm OS device's built in Memo application does.) DmArchiveRecord would be used to set an archive flag in the record to achieve this purpose.

17.1.1.2 Handling Synchronization Flags

During a synchronization all records that have been marked dirty (including new records) are retrieved and stored on the desktop. At the end of the synchronization these records are reset to not dirty. The dirty flag allows the synchronization process to quickly skip the unchanged records.

In some extreme cases the synchronization process must perform a slow sync process. In a slow sync, the desktop retrieves every record and compares it with its own records to determine changes since the last sync. Careful management of the dirty flags avoids this slower sync. Note the setting of the dirty flag in CreateRecord, DetailsApply, EditViewSelectCategory and EditViewSaveData.

Another flag to set when working with the desktop is the dmHdrAttrBackup flag in the database's header. StartApplication has been revised to set this flag when the database is created. This flag causes the database to be backed up to the desktop if there is no conduit specifically assigned to synchronize the database. This can be useful for Palm OS device data for which a developer does not want to create a conduit, like the high scores to a game. The Palm OS Desktop software can be used to restore the database back to the Palm OS device if necessary.

17.1.1.3 Handling Synchronization Related Launch Codes and More

During the synchronization process, when all records have been transferred between a Palm OS database and the desktop conduit, the application that created (owns) the database is sent a syncNotify launch command. This is a time for the application to perform any work needed to handle changes to its database. One common action is to resort the database as is done by the Palm OS device's built in Address Book. Another example, is that the Palm OS device's Datebook reschedules the pending alarms. This MemoPad application has no such work to perform.

The initDatabase launch command can come to an application to cause it to initialize its own database. This is especially useful for getting the database's appInfo block created with proper default data.

Other side effects of synchronization must be also be handled. In StartApplication the application's preferences must be checked more thoroughly to ensure that the saved values for current category, current record and scroll location are all still valid. A synchronization may have occurred and changed the database since the preferences were saved.

StartApplication also now handles the case where the entire database has been restored from the desktop without a valid appInfo block.

17.2 Building and Exercising the Application

With resources and code in place for Memo Pad 17, build the project and exercise it in the Pilot Simulator; follow these steps:

1. With CodeWarrior running, check the Project menu to make sure that the debugger is disabled before running the application. It should show the item Enable Debugger.
2. Build the application (CMD-M).
3. Launch the application (CMD-R). The Pilot Simulator starts and displays the Pilot with your memo pad application in it.

Verifying the features added in this phase is beyond the scope of this tutorial as it requires advanced setup and testing facilities. Some aspects of this phase, in particular the deleting of records, can be viewed with the Simulator's console window which is discussed in the next phase

Adding Shell Commands

Overview

In Phase 18, you add application specific “shell” commands and make them available in the Simulator’s Console window. Such commands are generally used for testing and debugging purposes such as to inspect and change the program's state interactively during run time. See Figure 18-1.

To work this phase, follow these steps:

1. If MemoPadMain.rsrc, MemoPadInfo.rsrc, MemoPadEdit.rsrc and MemoPadDetails.rsrc do not already exist in Working Folder:Rsc from Phase 17, then copy them from the MemoPad 17:Rsc folder.
2. Copy MemoPad.c, MemoPadRsc.c, and MemoPadRsc.h from the MemoPad 18:Src folder to Working Folder:Src.
3. If MemoPad.µ does not exist in Working Folder from Phase 17, copy it from MemoPad 17.

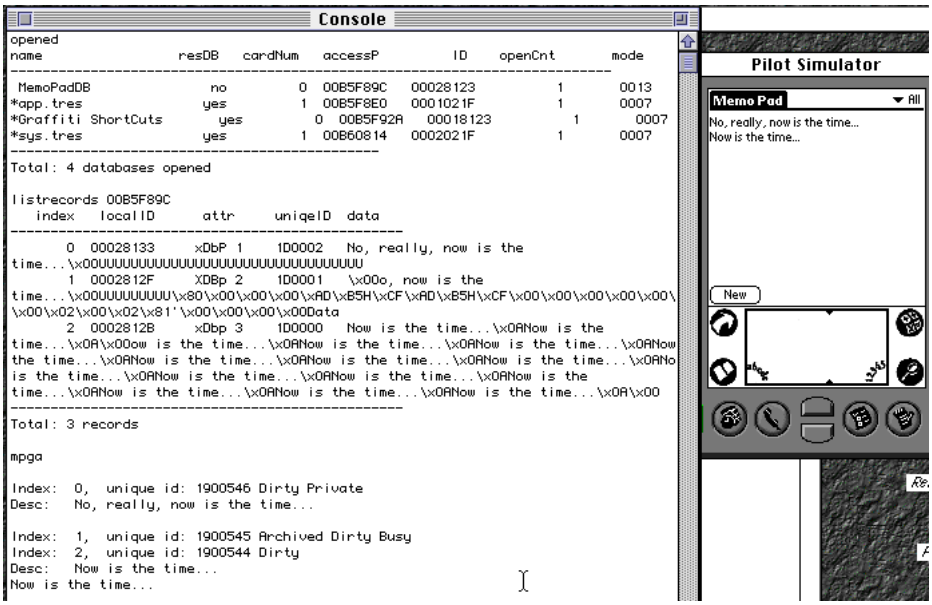

Figure 18.1 Phase 18 Memo Pad Console with New Shell Command.

No new resources are necessary for this phase.

18.1 Adding Console Commands

A custom console command can be an invaluable tool for testing or debugging your application in an efficient way. Common testing uses include populating a database with data or simulating the effects of a HotSync.

Debugging uses include customized displays of applications data or customized editing of data "behind the scenes." Spending time creating custom console commands will usually save many times as much debugging and testing time.

18.1.1 The Simulator Console Window

The Pilot Simulator has a Console window in which many commands are available. The behavior of the console window is similar to the MPW console or the MSDOS command line. These Simulator commands, called shell commands, have many uses.

Generally, they are used to provide a view into the Palm OS system that exists within the Simulator application. For example, memory cards and their databases can be inspected or changed; system Preference settings like sound can be changed, and some commands for the Simulator are provided like saving a log of the console window.

To use the commands, follow these steps:

1. Run the Pilot Simulator and select the Window>Console... menu item.
2. To see which databases are opened, type ‘opened’ and command-return (or press the enter key). Command-return needs to be typed whenever you want a line executed. The return key by itself simply inserts a new line.

   The opened command lists the databases along with their access pointer under the heading accessP. The access pointer is how Palm OS refers to the database.
3. List the contents of the MemoPad database (assuming you have created some records) by typing “listrecords” followed by the MemoPad database's access pointer. All the memos should be displayed

While the Simulator’s built in shell commands allow you to see the contents of databases, the process isn't an ideal one. It would be nice if the listrecords command would know about the MemoPad's database and be smart enough to find it instead of requiring an access pointer.

In addition, the presentation of the data isn't ideal. One really can't tell if the record is dirty or hidden. Also, notice that any multi-line records are listed only on one row.

The listrecords command is even less useful when the records are composed of other, non ASCII data. What is needed is a command tailored to how our application formats a record.

18.1.2 Creating a New Console Command

The Pilot Simulator has a hook to which unknown shell commands are passed. In the Simulator section of the MemoPad project file, is an entry for ShellCmdApp.cp. Open it up and look at it. It contains a dummy handler for unknown commands, ShlDoAppCmd, and a simple example of a shell command, DoAppCommand.

You will replace this file with one created specifically for the MemoPad application. You will have your own ShlDoAppCmd routine plus new routines to process MemoPad shell commands.

You will want to build our application with a local file (in the MemoPad folder), to handle your application-specific shell commands, rather than the generic ShellCmdApp.cp file.

To begin, from the MemoPad project file open the ShellCmdApp.cp:

1. Expand the Simulator section in the project files window by tapping the triangle in the left column. Tap the section name "Simulator" to select it.
2. Open the ShellCmdApp.cp file and select and copy all of the text in the file.
3. Create a new window by selecting the File:New menu item. Paste in the text from the clipboard.
4. Save the new file by selecting File:Save As into the Working Folder:Src folder with the file name ShellCmdMemoPad.cp.
5. Add the new file to the MemoPad project by selecting the Project:Add Window menu item. The entry for the new project file will be placed in the active section of the project window, Simulator. You can move the entry to another section by clicking and dragging on the file name.
6. Remove the entry for ShellCmdApp.cp from the project file by selecting the file in the project window and then selecting the Project:Remove Files menu item. Now, the project is using the local file for shell commands for the MemoPad application.

At this point you can add whatever shell commands you want to ShellCmdMemoPad.cp. To get the completed version of this file copy ShellCmdMemoPad.cp from the MemoPad 18:Src folder to Working Folder:Src. To view the code, from CodeWarrior; open ShellCmdMemoPad.cp.

A command to display the Memo Pad's records is what you want most. Working from the ShlDoAppCmd, you create the routine DoMemoPadGetAll. It takes one optional console window argument, the access pointer to the memo pad database.

Next, you create a routine called PrintMemoPadRecord that prints a single memo pad record given the database and an index number. It prints the unique id, the various attributes, and the memo itself. We

have DoMemoPadGetAll loop through all the records calling PrintMemoPadRecord for each one.

Lastly, finding the memo pad's access pointer is tedious, so create the routine FindOpenedMemoPadDatabase. It searches through every database for one created by the memo pad application and of a type that is used by the database. The type match is important so that the application itself, a database of type “appl”, isn't matched instead of the data database. DoMemoPadGetAll calls FindOpenedMemoPadDatabase when the user doesn't specify a database access pointer to use. Finish DoMemoPadGetAll by filling in the help for the command.

Now that DoMemoPadGetAll is finished, add it to the list in ShlDoAppCmd.

The list has three columns:

- The long name of the command.
- The abbreviated name for the same command.
- The routine to run when the command is entered in the console window.

As more custom commands for the MemoPad are created they will be added to this list and made available to the Console window.

When the memo pad is run with data, type “MemoPadGetAll”, or “mpga”, to see the contents of the database displayed in an easy to read presentation!

18.2 Building and Exercising the Application

With resources and code in place for Memo Pad 18, build the project and exercise it in the Pilot Simulator; follow these steps:

1. With CodeWarrior running, check the Project menu to make sure that the debugger is disabled before running the application. It should show the item Enable Debugger.
2. Build the application (CMD-M).
3. Launch the application (CMD-R). The Pilot Simulator starts and displays the Pilot with your memo pad application in it.

With the application up and running, exercise the features added in this phase; follow these steps:

1. Tap the New button on the main form to cause the edit form to be displayed. Enter a few words into the text field.
2. Create a few more records and make at least one of the private via the Details dialog.
3. Delete one of the records by using the Page>Delete Page menu command from the MemoPad's edit view.
4. Select the Simulator's Window:Console... menu item to open the Console window.
5. Enter the new shell command "MemoPadGetAll", or "mpga" and press the enter key or command-return. A formatted list of the records in the database will be displayed. Note that the attributes indicate the private and deleted (archived) records.
6. Quit the application; choose the Simulator File>Quit item.

Localizing For Other Countries

Overview

In Phase 19, you revise the process of building the application to allow the application to be localized for countries that use the French or German language. All of the displayed text is contained in the application's resource files. This text is translated into French and German and stored in language specific copies of the resource files. The build of the application is directed to choose a set of languages.

To work this phase, follow these steps:

1. If MemoPadMain.rsrc, MemoPadInfo.rsrc, MemoPadEdit.rsrc and MemoPadDetails.rsrc do not already exist in Working Folder:Rsc from Phase 18, then copy them from the MemoPad 18:Rsc folder.
2. Copy MemoPad.c, MemoPadRsc.c, MemoPadRsc.h, and ShellCmdMemoPad.cp from the MemoPad 19:Src folder to Working Folder:Src.
3. Copy the folder Rsc:French from the MemoPad 19:Rsc folder to Working Folder:Rsc.
4. Copy AppBuildRules.h from the MemoPad 19 folder to Working Folder.
5. If MemoPad.µ does not exist in Working Folder from Phase 18, copy it from MemoPad 18.

Figure 19.1 Phase 19 Memo Pad with French Localization.

No resource changes are necessary for this phase.

19.1 Localizing Your Application

The Palm OS SDK build environment contains the facility for building and maintaining your application for several languages simultaneously.

At this point in your development, you will profit greatly from the clean separation of the localizable and the non-localizable portions of your program that is afforded by the use of separate resource files. See the Chapter 4, “Designing, Testing, and Localization,” of the “Palm OS Cookbook” for more information.

Careful observation of the system-wide preference settings for dates, times, numbers, etc. will also help give your application a polished look when it is used in other languages.

19.1.1 Localization Techniques

Palm OS device is an international device. Designed to operate in many countries around the world, it is available in different languages and it can be customized for local conventions. Making Palm OS applications operate internationally with the Palm OS device is not difficult.

There are two types of localization efforts that may be done. The first is to localize the resources used by the application. Text should be translated into the desired language using either ResEdit or a similar resource editor. Sometimes the position or size of user interface components needs to be adjusted due to the length of translated strings.

Ideally the text is converted, possibly repositioned and no code changes are required to obtain a localized application. This can be achieved if localization is anticipated in the design and implementation of the application code.

A good example of such design is the MemoEdit title string. The template for the title string is "Memo # of #". A translator can convert the string and position the '#' characters where they belong for the new language. A change to the application code has been avoided.

It is a Palm OS development convention to store the resource files in a sub folder of Rsc which is named for the language. A French version of the application's resources appears in a French folder within the Rsc folder.

The second type of localization effort involves changing the code based on the country define. The same defines used in AppBuildRules.h can be used to conditionally compile the application's code.

An alternative to conditionally compiling the code is to check the country setting in the system preferences (defined in Preferences.h). This second method is more flexible but requires more code space.

19.1.2 Localization of MemoPad

To indicate to the Pilot Simulator that a different language is being used a local version of AppBuildRules.h is used. The AppBuildRules.h in the Incs folder is copied to the application folder.

Next, defines indicating the language and country settings are added to the local copy of AppBuildRules.h. The appropriate defines to be used are listed in Incs:BuildRules.h.

In addition, the MemoPadRsc.c file is modified to specify different directories for the application resource files based on the language define.

19.1.2.1 Local Version of AppBuildRules.h

To view the code, from CodeWarrior; open AppBuildRules.h. At the end of the file two defines have been added to set the language to French and the country to France.

These defines will be used to direct the build process and to direct conditional compilation of the source code. We want the project to find the local copy of AppBuildRules.h so select the CodeWarrior menu command Project:Reset File Paths.

19.1.2.2 Localized Resources

The Rsc:French folder contains a copy of the English version of MemoPad's resources. Several example text strings in these resources have been translated to French.

The MemoPadRsc.c file has been modified to determine the setting of the language define and conditionally include the application resources from either Rsc, Rsc:French or Rsc:German folder.

There are no code changes to the MemoPad application to support a French version.

Notice that the application resources are in French but the system resources, such as Find and CategoryEdit, are not. To have French system resources, copy the sys.tres file in Libraries:PalmOS:French over the version in Libraries:PalmOS.

Notice there is a German version as well. Restart the Simulator to use the translated system resources. You can restore the English sys.tres file from the Libraries:PalmOS:English folder.

19.2 Building and Exercising the Application

With resources and code in place for Memo Pad 19, build the project and exercise it in the Pilot Simulator; follow these steps:

1. With CodeWarrior running, check the Project menu to make sure that the debugger is disabled before running the application. It should show the item Enable Debugger.
2. Build the application (CMD-M).
3. Launch the application (CMD-R). The Pilot Simulator starts and displays the Pilot with your memo pad application in it.

With the application up and running, exercise the features added in this phase; follow these steps:

1. Note that the title of the main form and the New button have been localized.
2. Tap the New button on the main form to cause the edit form to be displayed. Note that the title and the Done and Details buttons have been localized.
3. Tap the Details button to display the details dialog. Note that the labels and buttons have been localized.
4. Quit the application; choose the Simulator File>Quit item.

Running the Application on Pilot

Overview

In Phase 20, you build the MemoPad application for execution on Pilot. This involves creation of a Makefile and using MPW to compile a Pilot executable. The Pilot Debugger is used to transfer the executable to the Pilot where it can be run.

To work this phase, follow these steps:

1. If MemoPadMain.rsrc, MemoPadInfo.rsrc, MemoPadEdit.rsrc and MemoPadDetails.rsrc do not already exist in Working Folder:Rsc from Phase 19, then copy them from the MemoPad 19:Rsc folder.
2. Copy MemoPad.c, MemoPadRsc.c, MemoPadRsc.h and MemoPad.r from the MemoPad 20:Src folder to Working Folder:Src.
3. Copy Makefile from the MemoPad 20 folder to Working Folder.
4. If MemoPad.µ does not exist in Working Folder from Phase 19, copy it from MemoPad 19.

No new resources are necessary for this phase.

20.1 Building an Executable for Pilot

Applications are built for the Palm OS device by executing a makefile from the MPW command line. This process will utilize the same core of files that we used to build the Simulator application. We will substitute a makefile for the CodeWarrior project file, and an <appname>.r file for the <appname>Rsc.c file.

The makefile and .r file used for this phase make excellent starting points for your own future projects.

20.1.1 Creating MemoPad.r

We begin by creating a MemoPad.r file for the application.
To view the code, from CodeWarrior; open Src:MemoPad.r.

NOTE: This file is not part of the project since it is not needed when running under the simulator.

This file describes to the Make process the resources used by the application when it is run on Pilot. This file is used in place of the MemoPadRsc.c file which is used when the application is built for the Simulator.

In the middle of the MemoPad.r file is a list which includes the user interface resources used by MemoPad. Prior to the list the code and data resources are described for the MemoPad.

The remaining settings are good general defaults to use for device applications.

20.1.2 Creating the Makefile and Building MemoPad for Pilot

Next, we create the Makefile; follow these steps:

1. To view the code, from CodeWarrior; open Makefile.

NOTE: This file is not part of the project since it is not part of the Simulator application.

---

2. Each .c file, other than your <appname>Rsc.c file, needs to be added to two places:
   a. First in the Compiles section, compilation instructions need to be added to this section for each .c file. The MemoPad application has only one file - MemoPad.c.
   b. Second, each resulting object file must be listed in the Objects section.
3. The Final Link section in the Makefile must be setup for the application.
   This section includes references to some settings in the MemoPad.r file. It also sets the application's creator identifier. It

should be 'Memo' to match the MemoPadAppType that is defined in MemoPad.c.

With both the Makefile and MemoPad.r filled out, we're ready to build the application; follow these steps:

1. Start up MPW.

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

---

2. In the Directory menu use Set Directory to set the directory to the Working Folder folder (where the Makefile resides).
3. Build the executable using the Build menu's build command. A successful make copies the executable to the Pilot Debugger directory.

It is also possible to make versions for other countries and in different languages. You do this by changing the COUNTRY and LANGUAGE options in the Makefile's C_OPTIONS section. The appropriate numbers are listed in Incs:BuildRules.h.

20.1.3 Sending the MemoPad Application to Pilot

To send the MemoPad application to Pilot; follow these steps:

1. Start the Pilot Debugger on the Mac.
2. Select the Console window (on the right) and drag it into full view. Make sure it has the focus.

NOTE: There are two principal windows in the Pilot Debugger; the Debugger window and the Console window. For our purposes, all work will be done in the Console window.

---

3. Place a Pilot in a cradle connected to the serial port of the Mac.
4. Launch the Preference application on the Pilot.

5. The first time this is done after a reset of the device, the Console window will show the message 'Ready...'
6. Write the shortcut symbol, a period (two dots), and then a '2'.
   Make sure the two is written in the number side of the Graffiti box. This sequence tells the device to listen on the serial connection to the Pilot Debugger on the Mac.
7. In the Console window type 'dir 0' and press the Enter key (NOT the Return key).
   You should see a list of the contents of the Pilot's memory card.
   If not, then the Pilot Debugger is not successfully communicating with the device. If necessary, the shortcut dot dot 2 sequence should be repeated.
8. To send the MemoPad application to the Pilot type 'import 0 Memopad.prc' in the Pilot Debugger's Console window.
   The '0' (zero) causes the application to be sent to the first memory card. Memopad.prc is the name of the executable to send. It was placed in the Debugger's directory by the makefile.
   When the import is complete, the MemoPad application will show up in the device's launcher window.
9. When you send other versions of the MemoPad to the Pilot, any existing versions must be removed first. The Pilot Debugger's Console window command 'del 0 MemoPad' accomplishes the removal.
10. When you are done using the Pilot Debugger it is necessary to reset the Pilot. (The reset hole is on the back of the device.)

NOTE: Until this is done the serial port of the Pilot is held by the remote debug nub waiting for more commands from Pilot Debugger. This means that HotSync will fail and that the power is continually fed to the serial port from the batteries.

You are now ready to compile and try out any Pilot application you want to create!

Pilot Resource Cookbook

This resource cookbook gives you a place to look up the implementation of any of Pilot's resources used in the tutorial. Resources are listed in alphabetic order. For more detailed information, see Chapter 3, "Palm OS User Interface Resources," of "Developing Palm OS Applications, Part I."

Applications may use any resource IDs less than 10,000. The system reserves resource IDs 10,000 and greater. Before setting the attributes in any of these resources, you must have already:

- Started ResEdit.
- Opened a .rsrc file.
- Created a new resource, i.e., chosen Resource>Create New Resource (CMD-K).
- Entered the resource type ("tFRM" for a form resource) in the Select New Type dialog and clicked OK.

ICON

ICON is the image that appears in the launcher. Only the top 20 lines of the 32 available lines are used. This is a Macintosh resource, not a Palm OS resource.

MBAR

MBAR creates a menu bar, a container used to group menus (see the MENU resource item). This is a Macintosh resource, not a Palm OS resource.

- **# of menus**. ResEdit increments this item as you add menus to the menu bar.
- 1) ****. Each of these numerical entries represents a menu item.

Here's how to add a menu item:

1. In the editor window, click the number item like "1) ******" (or the next open number in the series).
2. Choose Resource>Insert New Field(s) (CMD-K).

3. Set the menu resource ID (Menu res ID) as appropriate to your resource numbering scheme. Eventually, you add the resource ID here for each menu belonging to the particular menu bar.
4. Set resource Info; choose Resource>Get Resource Info (CMD-I).
5. In the Info window, set the ID and enter a name for the menu.

MENU

MENU creates a menu, its items, and subitems. This is a Macintosh resource, not a Palm OS resource.

- **Enabled**. Check this box to enable this menu.
- **Title**. When you first open a menu resource, the default title is for a menu, not for a menu item. Once you've entered a menu name (like "File,"), you enter its items ("New," "Open," and so on). **Note**: For the first item, type a name. The Apple item and subitems do not apply to Pilot.
  Once you enter a menu title, a Text field appears. Type the desired menu item name in this field.
- **Color**. This item does not apply to Pilot.
- **Cmd-Key**. This corresponds to the command stroke combination on the Pilot. Type the command letter in this field and it appears (with the propeller symbol) on the same line with the item name.
- **Mark**. This does not apply to Pilot.
- **Menu Ids**. A menu ID must be set for each menu item so your program can tell which item a user selects. Use the menu command Menu>Edit Menu to pick an ID for the first item in the menu. Each subsequent item will use an ID one higher than the prior. Note that separators are also assigned a menu ID. These menu IDs should match those defined in the program's resource header file.

PICT

PICT is a resource for including PICT-format graphical files in Pilot applications. To use it, you must copy the desired graphic onto the system clipboard. After opening the PICT resource, paste in the graphic from the clipboard. This is a Macintosh resource, not a Palm OS resource.

To set resource info for your PICT resource; follow these steps:

1. Set resource Info; choose Resource>Get Resource Info (CMD-I).
2. In the Info window, set the ID and the name.

tAIN

tAIN is the application icon name resource. Provides a name for the application in the launcher.

- **App Icon Name**. The name of the application. Should be short and not conflict with other applications on the launcher.

Talt

Talt is the alert dialog resource. **Note:** Unlike other Palm OS resources, Talt begins with a capital "T."

- **Alert Type**. The alert type determines which of four possible icons is displayed in the alert and the sounds that it plays when the alert is drawn. See the Alert Type table (Table 2) at the end of the cookbook for a description of the four different options.
- **Help Res ID**. The resource ID of a string resource (tSTR) that is the help text for the alert dialog box.
- **# Buttons**. The number of buttons the alert contains.
- **Default Button**. The ID defining which button activates, when the user switches to another application.
- **Title**. The title displayed on the alert dialog.
- **Message**. The message displayed on the alert dialog.
- "1) ******". These fields contain the text of the buttons displayed by the alert. There is an entry in the resource for each button. The placeholder for the first entry is indicated by the label "1) *****".

Here's how to add a button to the alert dialog:

1. In the editor window, click the number item like "1) ******" (or the next open number in the series).
2. Choose Resource>Insert New Field(s) (CMD-K).
3. Enter the button name.

4. Set resource Info; choose Resource>Get Resource Info (CMD-I).
5. In the Info window, set the ID and enter a name for the button.

tBTN

tBTN is the button resource.

- **Button ID**. The ID number should reflect the number of the form containing the button.
- **Left**. The number of pixels to the left edge of the container resource.
- **Top**. The number of pixels to the top of the container resource.
- **Width**. The width of the button in pixels.
- **Height**. The height of the button in pixels.
- **Usable**. True sets the button to be active, visible, and a part of the interface. A non-usable object is not part of the interface and is not drawn. However, you can programmatically reset non-usable to usable.
- **Left Anchor**. This attribute controls how the button resizes itself when its text label is changed. If the attribute is true, then the left bound of the button is fixed, if false the right bound is fixed.
- **Frame**. True gives the button a frame.
- **Non-bold.** True gives the button a single pixel width frame. False creates a bold frame.
- **Font**. This number sets the button's Label font. See the Font ID table (Table 1) at the end of this cookbook.
- **Label**. The caption that appears on the button.

tCBX

tCBX is the check box resource.

- **Check Box ID**. This is the resource ID for the particular check box (relative to its container resource).
- **Left**. The number of pixels to the left edge of the container resource.
- **Top**. The number of pixels to the top of the container resource.

- **Width**. The width of the check box in pixels.
- **Height**. The height of the check box in pixels.
- **Usable**. True sets the check box to be checkable and visible. A non-usable object is not part of the interface and is not drawn. However, you can programmatically reset non-usable to usable.
- **Selected**. True means checked and false not checked.
- **Group**. The group number for the check box. Grouped check boxes are mutually exclusive. Setting the group is necessary to support this mutual behavior. Code is needed to actually enforce the behavior.
- **Font**. Choose from among fonts 0 through 2. See the Font ID table (Table 1) at the end of the cookbook.
- **Label**. The text displayed in the check box.

tFBM

tFBM is the form bitmap resource. These bitmaps are used in Alert dialogs.

- **X Position**. The number of pixels to the left edge of the container resource.
- **y Position**. The number of pixels to the top of the container resource.
- **Bitmap Res ID**. Set this to the ID of a PICT resource to use.
- **Usable**. True sets the bitmap to be visible. A non-usable object is not part of the interface and is not drawn. However, you can programmatically reset non-usable to usable.

To set resource info for your form bitmap resource; follow these steps:

1. Set resource Info; choose Resource>Get Resource Info (CMD-I).
2. In the Info window, set the ID and the name.

tFLD

tFLD is the field resource. It can display one or more lines of text.

- **Field ID**. This is the resource ID for the particular field (relative to its container resource).

- **Left**. The number of pixels to the left edge of the container resource.
- **Top**. The number of pixels to the top of the container resource.
- **Width**. The width of the field in pixels.
- **Height**. The height of the field in pixels.
- **Usable**. True makes the field visible. A non-usable object is not part of the interface and is not drawn. However, you can programmatically reset non-usable to usable.
- **Editable**. True means the field is editable. A non-editable field won't accept text.
- **Single Line**. True means the field is not horizontally scrollable, won't accept tab or carriage return characters. Only a single line is displayed.
- **Dynamic Size**. True means that field height is expanded or contracted as characters are added or removed (dynamic sizing). Set this to False if **Single Line** is True.
- **Left Justified**. True sets text to left margin. Supported only when **Single Line** is True.
- **Max Chars**. Specifies the maximum number characters allowed in the field. (32767 is the maximum.)
- **Font ID**. Choose from among fonts 0 through 2. See the Font ID table (Table 1) at the end of the cookbook.

tFRM

tFRM is the form resource. A form acts as a container for other resources.

- **Left**. The number of pixels to the left edge of the Pilot display.
- **Top**. The number of pixels to the top of the Pilot display.
- **Width**. The width of the form in pixels.
- **Height**. The height of the form in pixels.

Ignore "invalid rectangle" messages. ResEdit does not like bounds where the fourth parameter is less than the second.

  - **Set** button. Ignore this; it does not apply to Pilot.
- **Useable**. Not currently supported.

- **Modal**. True processes pen events in the current form only. Other parts of the interface become inaccessible. Generally, this option is used with dialog boxes.
- **Save behind**. True saves pixels behind the form and restores them, when the form goes away. Generally, this option is used with dialog boxes.
- **Number of Objects**. A counter that ResEdit increments as objects are added to the form.
- **Form**. Each form must have its own ID number. Recommendation: Start at 1000 and increment by 100.
- **Help Rsc**. The number of a tSTR resource to use for this form. A non zero value places an information icon at the top right of the form. When the icon is pressed a dialog appears displaying the string. This feature should be used in dialogs only.
- **Menu Rsc**. The number of a MBAR resource to use for this form.
- **Default Button ID**. The number of a button to assume is pressed when the form is forced to go away (by switching to another application).

tGDT

tGDT is the gadget resource. It indicates that a particular region of the form is handled by the application. This is how applications make custom ui components.

- **Gadget ID**. This is the resource ID for the gadget.
- **Left**. The number of pixels to the left edge of the container resource.
- **Top**. The number of pixels to the top of the container resource.
- **Width**. The width of the gadget in pixels.
- **Height**. The height of the gadget in pixels.
- **Usable**. True makes the gadget visible. A non-usable object is not part of the interface and is not drawn. However, you can programmatically reset non-usable to usable.

tGSI

tGSI specifies where to display the Graffiti shift state indicator. These bitmaps are used in Alert dialogs.

- **x pos**. The number of pixels to the left edge of the container resource.
- **y pos**. The number of pixels to the top of the container resource.

States include punctuation, symbol, uppercase shift, and uppercase lock.

tLBL

tLBL is the label resource.

- **Label ID**. This is the resource ID for the particular label (relative to its container resource).
- **Left**. The number of pixels to the left edge of the container resource.
- **Top**. The number of pixels to the top of the container resource.
- **Usable**. True sets the form to be visible and false not visible. A non-usable object is not part of the interface and is not drawn. However, you can programmatically reset non-usable to usable.
- **Font ID**. See the Font ID table (Table 1) at the end of the cookbook.
- **Text**. String displayed in the label.

tLST

tLST is the list resource.

- **List ID**. This is the resource ID for the particular list (relative to its container resource).
- **Left**. The number of pixels to the left edge of the container resource.
- **Top**. The number of pixels to the top of the container resource.
- **Width**. The width of a list item in pixels.
- **Usable**. True sets the form to be visible and false not visible. A non-usable object is not part of the interface and is not

drawn. However, you can programmatically reset non-usable to usable.

- **Font**. Choose from among fonts 0 through 2. See the Font ID table (Table 1) at the end of the cookbook.
- **Visible Items**. Height of list (in number of visible items). If you enter zero, the number of items (**Items**) is used.
- **Items**. The number of items (choices), incremented by the system as you add them. See below for how to add an item to the list.

To add a list item:

1. In the editor window, click the number item like "1) ******" (or the next open number in the series).
2. Choose Resource>Insert New Field(s) (CMD-K).
3. Enter the name of the desired list item in the Text field.

tPBN

tPBN is the push button resource. Push buttons usually indicate exclusive options.

- **Push Button ID**. The ID number should reflect the number of the form containing the push button. For example, if the resource is 1000, the Push Button ID can be anywhere between 1001 and 1099.
- **Left**. The number of pixels to the left edge of the container resource.
- **Top**. The number of pixels to the top of the container resource.
- **Width**. The width of the push button in pixels.
- **Height**. The height of the push button in pixels.
- **Usable**. True sets the push button to be active, visible, and a part of the interface. A non-usable object is not part of the interface and is not drawn. However, you can programmatically reset non-usable to usable.
- **Group**. The group number for the push button. Grouped push buttons are mutually exclusive.
- **Font**. This number sets the push button's Label font. See the Font ID table (Table 1) at the end of this cookbook.

- **Label**. The caption that appears on the push button.

tPUL

tPUL is the pop up list resource. These are used to associate a pop up trigger with a list to pop up.

- **Control ID**. The ID number of the popup trigger.
- **List ID**. The ID number of the list.

tPUT

tPUT is the pop up trigger resource. These are used to pop up a list for the user to select from.

- **Trigger ID**. The ID number should reflect the number of the form containing the trigger. For example, if the resource is 1000, the Trigger ID should be anywhere between 1001 and 1099.
- **Left**. The number of pixels to the left edge of the container resource.
- **Top**. The number of pixels to the top of the container resource.
- **Width**. The width of the trigger in pixels.
- **Height**. The height of the trigger in pixels.
- **Usable**. True sets the trigger to be active, visible, and a part of the interface. A non-usable object is not part of the interface and is not drawn. However, you can programmatically reset non-usable to usable.
- **Left Anchor**. True fixes the left bound of the trigger, when the size of the label changes.
- **Frame**. True gives the trigger a frame.
- **Non-bold.** True gives the button a plain frame. False creates a bold frame.
- **Font**. This number sets the trigger's Label font. See the Font ID table (Table 1) at the end of this cookbook.
- **Label**. The caption that appears on the trigger.

tREP

tREP is the repeat button resource. These are identical to buttons except they repeatedly send events while they are pushed.

- **Button ID**. The ID number should reflect the number of the form containing the repeat button. For example, if the resource is 1000, the Button ID can be anywhere between 1001 and 1099.
- **Left**. The number of pixels to the left edge of the container resource.
- **Top**. The number of pixels to the top of the container resource.
- **Width**. The width of the repeat button in pixels.
- **Height**. The height of the repeat button in pixels.
- **usable**. True sets the repeat button to be active, visible, and a part of the interface. A non-usable object is not part of the interface and is not drawn. However, you can programmatically reset non-usable to usable.
- **Left Anchor**. True fixes the left bound of the repeat button, when the size of the label changes.
- **Frame**. True gives the repeat button a frame.
- **Non-bold frame.** True gives the repeat button a plain frame. False creates a bold frame.
- **Font**. This number sets the repeat button's Label font. See the Font ID table (Table 1) at the end of this cookbook.
- **Label**. The caption that appears on the repeat button.

tSEL

tSEL is the selector resource. These indicate that a dialog will appear to edit the contents of the selector.

- **Selector Trigger ID**. The ID number should reflect the number of the form containing the selector trigger. For example, if the resource is 1000, the Selector Trigger ID can be anywhere between 1001 and 1099.
- **Left**. The number of pixels to the left edge of the container resource.
- **Top**. The number of pixels to the top of the container resource.
- **Width**. The width of the selector trigger in pixels.
- **Height**. The height of the selector trigger in pixels.
- **Usable**. True sets the selector trigger to be active, visible, and a part of the interface. A non-usable object is not part of the

interface and is not drawn. However, you can programmatically reset non-usable to usable.

- **Left Anchor**. True fixes the left bound of the selector trigger, when the size of the label changes.
- **Font**. This number sets the selector trigger's Label font. See the Font ID table (Table 1) at the end of this cookbook.
- **Label**. The caption that appears on the selector trigger.

tSTR

tSTR is the string resource.

- **The String**. A text string (in decimal ASCII).
- **Data $**. A text string in (hexadecimal ASCII).

NOTE: The string resource uses either field. If you enter data in both, they are concatenated.

To set resource info for your string resource; follow these steps:

1. Set resource Info; choose Resource>Get Resource Info (CMD-I).
2. In the Info window, set the ID and the name.

tTBL

tTBL is the table resource.

Table ID. This is the resource ID for the table.

- **Left**. The number of pixels to the left edge of the container resource.
- **Top**. The number of pixels to the top of the container resource.
- **Width**. The width of the table in pixels.
- **Height**. The height of the table in pixels.
- **Editable**. True means the table is editable. A non-editable table won't accept text.
- **Rows**. The number of rows in the table.
- **Columns**. The number of columns in the table, incremented by the system as you add them. See below for how to add a column to the table.

To add a table column:

1. In the editor window, click the number item like “1) ******” (or the next open number in the series).
2. Choose Resource>Insert New Field(s) (CMD-K).
3. Enter the width of the column in pixels.

tTTL

tTTL is the title resource. A title assigns a visible title to another resource.

Title. The title string.

After typing in a title, you need to assign an ID number and a name to this resource. To set resource Info; choose Resource>Get Resource Info (CMD-I) for the Macintosh.

---

tver

tver is the version string resource. Provides a version for the application.

VersionString. The version of the application.

Figure 20.1 Font ID’s

FontID	Name
0	9 pt plain font
1	9 pt bold font
2	12 pt plain font
3	Symbol font
4	Check box font

FontID	Name
5	Symbol font
6	Calculator font

Table 20.1 Alert Type

Alert Number	Type
0	informationAlert
1	confirmationAlert
2	warningAlert
3	errorAlert