CodeWarrior®

Palm OS 2.0 Tutorial for Windows

Because of last-minute changes to CodeWarrior, some of the information in this manual may be inaccurate. Please read the Release Notes on the CodeWarrior CD for the latest up-to-date information.

Revised: 98/01/19 map

Metrowerks CodeWarrior copyright ©1993–1998 by Metrowerks Inc. and its licensors. All rights reserved.

Documentation stored on the compact disk(s) may be printed by licensee for personal use. Except for the foregoing, no part of this documentation may be reproduced or transmitted in any form by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without permission in writing from Metrowerks Inc.

Metrowerks, the Metrowerks logo, CodeWarrior, and Software at Work are registered trademarks of Metrowerks Inc. PowerPlant and PowerPlant Constructor are trademarks of Metrowerks Inc.

All other trademarks and registered trademarks are the property of their respective owners.

ALL SOFTWARE AND DOCUMENTATION ON THE COMPACT DISK(S) ARE SUBJECT TO THE LICENSE AGREEMENT IN THE CD BOOKLET.

How to Contact Metrowerks:

U.S.A. and international	Metrowerks Corporation P.O. Box 334 Austin, TX 78758 U.S.A.
Canada	Metrowerks Inc. 1500 du College, Suite 300 Ville St-Laurent, QC Canada H4L 5G6
Ordering	Voice: (800) 377–5416 Fax: (512) 873–4901
World Wide Web	`http://www.metrowerks.com`
Registration information	`register@metrowerks.com`
Technical support	`support@metrowerks.com`
Sales, marketing, & licensing	`sales@metrowerks.com`
CompuServe	goto `Metrowerks`

Table of Contents

Introduction

This tutorial guides developers through the process of creating and testing a sample Palm OS application. The sample application is a memo pad, similar to the built-in Memo Pad in a PalmPilot device. The tutorial teaches the basic techniques for:

- Creating a PalmPilot user interface using Constructor.
- Programming and debugging the PalmPilot device using the tools in the "CodeWarrior for PalmPilot" tool set.

The tutorial consists of phases. Each phase steps you through creating a fully functional application. The steps describe what to do and why. Each phase builds on and extends the capabilities of the previous phase. Source code for each completed phase is included, so that you can build each phase independently.

To work with the tutorial, you must have installed CodeWarrior for PalmPilot, including the Constructor tool (see Release Notes and installation instructions).

To develop software in this environment, you should be an experienced C or C++ programmer, familiar with the desktop environment (Macintosh or Windows) and its development tools. A review of "Application Control Flow Overview" in Chapter 2 of "Developing Palm OS Applications, Part 1" is highly recommended. For more detailed information about Constructor and CodeWarrior, see the CodeWarrior Documentation folder.

This introduction discusses the following topics:

System Requirements for Macintosh Users

This section specifies the hardware and software requirements for running the Palm OS SDK and tutorial on a Macintosh.

Hardware Requirements

- CPU: Macintosh Quadra 700 or higher; PowerPCs are fine.
- RAM: 16 MB minimum; 24 MB-3 2MB recommended
- Disk space: 95+ MB

Software Requirements

- Macintosh System Software 7.1 or later for 68030 or 68040 processors
- Macintosh System Software 7.1.2 or later for the PowerPC 601 and 604 processors
- CodeWarrior for PalmPilot

Note: If you're developing on the Macintosh, it's possible to use ResEdit to create the user interface resources.

System Requirements for Windows Users

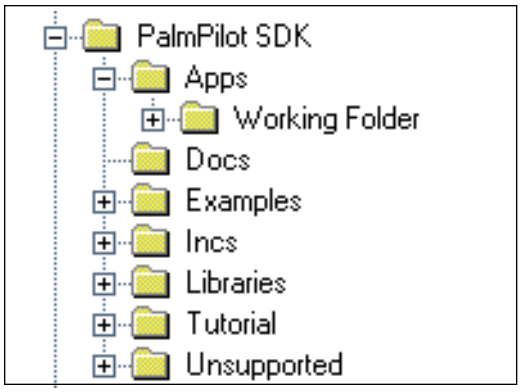
This section specifies the hardware and software requirements for running the Palm OS SDK and tutorial under Windows.

- PC running Windows 95 or Windows NT 4.0 or higher
- 24 MB of RAM
- 90 MB of free hard disk space

Navigating Tutorial Phases

Folder Structure

The following folder structure has been created to facilitate working with the tutorial:



- The `Tutorial` folder contains all the source code files for each phase. These serve as a backup and standard against which you can check your work.
- The `Working Folder`, located in the `Apps` folder, is where you should build each phase of this tutorial. Each subsequent phase builds on work you did in the previous phase.

Folder Setup

For each tutorial phase you create new resources, but examine a code file that's already prepared for you.

To set up work folders:

Copy...	From...	To...
`Code files (*.c)`	`Tutorial\Current phase\Src`	`Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\Previous phase\Src`	`Working Folder\Src`

Copy...	From...	To...
MemoPad.rsrc	Tutorial\Previous phase\ Src\Resource.frk	Working Folder\ Src\Resource.frk
MemoPad.mcp	Tutorial\Previous phase	Working Folder

Note: You'll only be able to see the .c or .rsrc files if the view option "Show DOS file extensions" is on.

Under Windows, you explicitly have to copy both the higher-level resource file and the one from the resource.frk directory. To open and edit the file, choose the one at the higher level and Constructor will update both.

For example, for Phase 3, follow these steps:

Copy these files...	From...	To...
MemoPad.c	Tutorial\MemoPad 03\Src	Working Folder\Src
MemoPad.rsrc	Tutorial\MemoPad 02\Src	Working Folder\Src
MemoPad.rsrc	Tutorial\MemoPad 02\Src\ Resource.frk	Working Folder\Src\ Resource.frk
MemoPad.mcp	Tutorial:MemoPad 02	Tutorial

Note: If you work through the phases sequentially, you don't have to move either of the `.rsrc` files and the `MemoPad.mcp` file.

Working With the Tutorial

It's a good idea to work through all Tutorial phases to get the best sense of how an application is built. The phases are set up cumula-

tively, so that the resource files in each succeeding phase incorporate the changes made in the previous phase. For each phase, you will:

- Revise the UI resource file to reflect the interface for that phase.
- Examine the already completed source code.

Because a completed version of each phase is provided, you can work with the tutorial in two ways:

- Go through the phases sequentially without copying any files from the tutorial source after phase 0.
- Work with individual phases, using the setup instructions provided in each phase.

You can also copy a complete phase folder to the Working Folder and build it to produce the same result as working through the phase.

Resetting the Device

As you download your application on the device and debug it there, you may have to reset your device. This may happen if the connection with the debugger doesn't work quite properly or if your application has a problem.

Any reset is normally performed by sticking a bent-open paper clip or a large embroidery needle into the small hole in the back of the PalmPilot device (the end of an earring also often works). This hole, known as the "reset switch" is in the middle of the serial number sticker. Depending on additional keys held down, the reset behavior varies. You can choose between:

- Soft Reset
- Soft Reset + Up Arrow
- Hard Reset

Soft Reset

A soft reset clears all of the dynamic heap (Heap 0, Card 0). The storage heaps remain untouched. The operating system restarts from scratch with a new stack, new global variables, restarted driv-

ers, and a reset comm port. All applications on the device receive a `SysAppLaunchCmdReset` message.

Soft Reset + Up Arrow

Holding the up-arrow down while pressing the reset switch with a paper clip causes a soft reset with the following two exceptions:

- The `SysAppLaunchCmdReset` message is not sent to applications. This is useful if there is an application on the device that crashes upon receiving this message (not uncommon) and therefore prevents the system from booting.
- The OS won't load any system patches during startup. This is useful if you have to delete or replace a system patch database. If the system patches are loaded and therefore open, they cannot be replaced or deleted from the system.

Hard Reset

A hard reset is performed by pressing the reset switch with a paper clip while holding down the power key. This has all the effects of the soft reset. In addition, the storage heaps are erased. As a result, all programs, data, patches, user information, etc. are lost. A confirmation message is displayed asking the user to confirm the deletion of all data.

The `SysAppLaunchCmdReset` message is sent to the application at this time. If the user selected the "Delete all data" option, the digitizer calibration screen comes up first. The default databases for the four main applications are copied out of the ROM.

If you hold down the up arrow key when the "Delete all data" message is displayed, and then press the other four application buttons while still holding the up arrow key, the system is booted without reading the default databases for the four main applications out of ROM.

Deleting the MemoPad Application

After you've downloaded the MemoPad application onto the device, you need to delete it if you want to download other versions. To do this, follow these steps:

1. **On the device, tap the "Applications" silk-screened button.**
2. **Choose Memory, then tap the "Delete apps" button.**
3. **On the screen that appears, select MemoPad, then the Done button.**

Note that MemoPad differs from Memo Pad, the application that's built into the ROM.

Creating a Form and a Button

Phase 1 differs from other phases of the tutorial in that you must create a project. Working through Phase 1 gives you the experience of creating all the pieces of a project except the source code.

Note: If you've never worked with the Metrowerks tools, going through this process is especially important. In addition, you should also work with the CodeWarrior documentation in the CodeWarrior Documentation folder.

When you begin creating your own projects, you can use Phase 1 as a model for fitting the pieces together. Note, however, that Phase 17 has the best starting files; this phase only illustrates the process.

The basic steps in Phase 1 are as follows:

- Creating a Resource File
- Components of a Palm OS Project
- Modifying a CodeWarrior Project
- Setting File Access Paths
- Components of a Palm OS Project
- Examining MemoPad.c
- Examining MemoPadRsc.h
- Building and Downloading the Application

Convention: Phase 1 explicitly instructs you to launch Constructor before you start working. Subsequent phases of the Tutorial assume that you have the tool running.

Project Setup

The Palm OS SDK contains a series of folders. To work with these folders, copy them as follows:

1. **Open the PalmPilot SDK folder.**
2. **Perform setup for Phase 1, as follows:**

Copy these files...	From...	To...
`MemoPad.c`	`Tutorial\MemoPad 01\Src`	`Apps\Working Folder\Src`
`MemoPad.mcp`	`Tutorial\MemoPad 00`	`Apps\Working Folder`

Note: Be sure to copy (Ctrl-drag), **not** move the files.

Creating a Resource File

This section explains how to create a UI resource file in the following sections:

Creating a New Resource File

When creating a Palm OS project, most application developers start by designing a UI prototype. This involves creating UI forms and their components and saving them in a resource file. (A resource is a statically initialized UI object or string.)

In this section, you'll create a new resource file and learn about the different windows that Constructor provides to let you create your UI.

How to Create a Resource File

To create a resource file for your project, follow these steps:

1. **Launch Constructor.**
2. **Select File > New Project file.**

   Constructor displays the Constructor project window to allow you to create resources for this project.

For background information about resources, read the next section. To continue with the tutorial steps, go to "Creating a Form With a Title" on page 26.

Resources and Project Settings

Constructor lets you create resources and change project settings:

- **Project resources** can be instantiated from a template in the top half of the Constructor project window using Ctrl-K.

  Project resources are associated with the project, but not necessarily with a specific form. There are Forms, Menu Bars, Menus, Strings, Alerts, Icons, and Bitmaps.

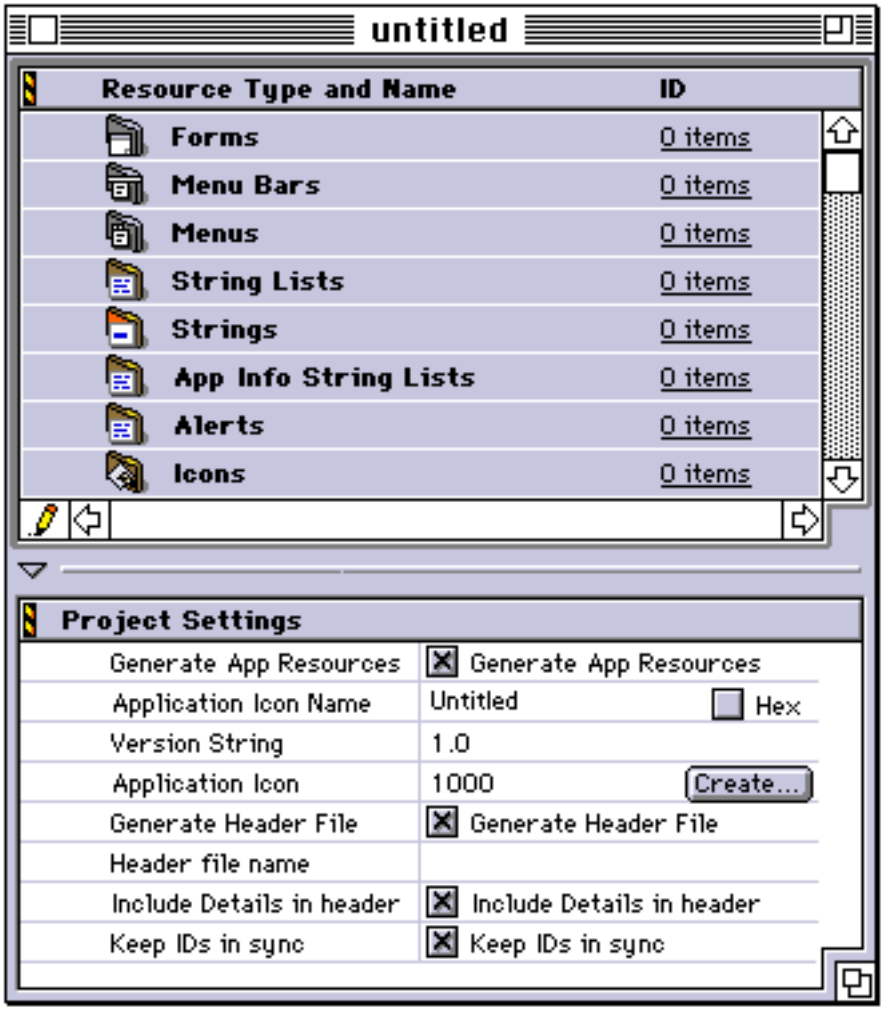
- **Catalog resources** are drag-and-drop UI objects that are part of a Form. You can view them by choosing Window > Catalog or Ctrl-Y.

  The Catalog's UI palette contains a Button, Checkbox, Field, Form Bitmap, Gadget, Graffiti Shift Indicator, Label, List, Popup Trigger, Push Button, Repeating Button, Scrollbar, Selector Trigger, and a Table.

- **Project settings** are associated with the project as a whole. They are not part of the Graphical User Interface but provide other important information about the project.

Both types of resources and the project settings are saved with the project.

The figure below shows a Constructor project window. The upper panel, Resource Type and Name, lists resource types you can create for your project, and the ID number of each type you create.



Tip: The attributes of each UI object are explained in some detail in “CodeWarrior Constructor for Palm OS,” which is part of the CodeWarrior document set. You can also refer to Chapter 3, “Palm OS UI Objects,” in Developing Palm OS Applications, Part I.

Creating a Form With a Title

In this section, you create a form with a title and a button.

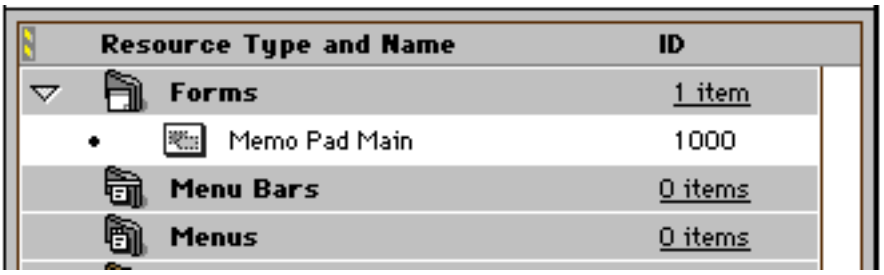
1. **In the Constructor project window, select "Forms".**
2. **Type Ctrl-K (or choose Edit > New Form Resource).**

   Constructor creates a new Form resource, which is displayed on the next line.

Tip: If the New Resource command (and others) is greyed out when you open the Edit menu, check the Pencil icon in the bottom-left corner of the Constructor project window. If there is a slash through it, the file is write protected (locked). Click the pencil to unlock the file and reactivate your menu commands.

---

3. **Rename the new "untitled" Form "Memo Pad Main" by clicking the name, then typing the new name.**



4. **Double-click the Memo Pad Main icon.**

   Constructor displays the Form Editor window. Each form has its own editor window in which you can interactively design the UI for this form by dragging catalog icons (which represent UI elements) onto it.

   You can also change the dimensions and origin of the form, associate a menu with it, and give it a title.

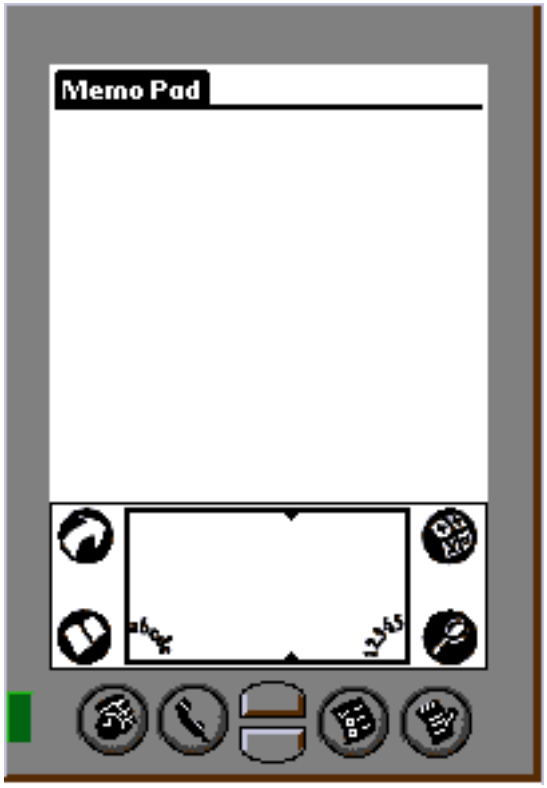
5. **Select the Layout Appearance Panel, then choose Layout > Hide Object IDs.**

   If the menu shows "Show Object IDs", leave it untouched.

   Hiding object IDs makes working with the Form Editor window easier at the initial design stage.

6. **In the Layout Properties panel (left half of the Form Editor window), click next to "Form Title" and type in "Memo Pad", then press Enter.**

   You now have a named Form with a title displayed in the Layout Appearance panel, as shown in the picture below:



Adding a Button to the Form

To create the first button for the MemoPad, follow these steps:

1. **If the Catalog window is not visible, type Ctrl-Y to open it.**
2. **Select the Button icon and drag it from the Catalog into the Layout Appearance panel (the right panel of the Form Editor window).**
3. **Set the button properties as follows. You can use the Tab key to move from one field to the next:**

Object Identifier	Exit
Button ID	(Assigned by Constructor)
Left and Top Origin	110, 146
Width and Height	40, 12
All checkboxes	Leave checked
Font	Choose Standard from the pop-up menu
Label	Exit

4. **In the Project Settings panel, the bottom half of the Constructor project window, set the Header file name to MemoPadRsc.h.**

   When you compile a project with the `MemoPad.rsrc` file, CodeWarrior now automatically generates a header file with unique names for the different resources. The names, which are based on the Object Identifier, are used to retrieve and manipulate the UI objects in the application source code.

5. **Save the resource file as Apps\Working Folder\Src\MemoPad.rsrc.**

Components of a Palm OS Project

Each Palm OS project usually has three types of files:

Source files	These files are in the `Src` folder. Each Palm OS application usually has at least three source files: • `MemoPad.c`—contains the C code for the application, including the event loop and key functions. • `MemoPadRsc.c`—identifies the set of application resource files that the application requires at runtime. • `MemoPadRsc.h`—contains `#define` macros for resource IDs that are used by the application. These macros map resource names to resource IDs. This file is automatically updated when you save the resource file in Constructor.
Resource files	This file is in the `Src` folder. It contains the user interface resources. There is usually only one resource file per project. You've just saved the resource file `MemoPad.rsrc`. Under Windows, there's also a `MemoPad.rsrc` file in the Resource.frk folder. This file is maintained internally by CodeWarrior. You don't have to concern yourself with it unless you copy projects; in that case, copy both `.rsrc` files. The application's resources currently consist of the Main form and its components. All resources are contained in the `MemoPad.rsrc` file you created earlier with Constructor. The set of required resources grows as you add more resources to the application.
Project files	Contain information on how to build the project.

Modifying a CodeWarrior Project

This section shows you how to modify a CodeWarrior project file. The Components of a Palm OS Project, including the project file, are discussed in the previous section.

Adding Files to the Project

To add source files to the project, follow these steps:

1. **Double-click on MemoPad.mcp in the Working Folder.**

   This loads Metrowerks CodeWarrior IDE with the project window `MemoPad.mcp` opened.

2. **In the project window, select the placeholder "replace me.c" in the Application Source area. (Expand Application Source if needed.)**

3. **With the placeholder still selected in the project window, choose Project > Add Files.**

4. **Navigate to the Working Folder:Src folder.**

5. **From the dialog box, double-click on the `MemoPad.c` source file, then click Add.**

   This adds the source file to the project.

6. **For this first phase, you also have to remove "replace me.c" by first selecting the file, then choosing Project > Remove Selected Items.**

Tip: Instead of using the menu command, you can also drag files directly onto the window. To remove a file, select it and press Ctrl–Delete.

Adding Resource Files to the Project

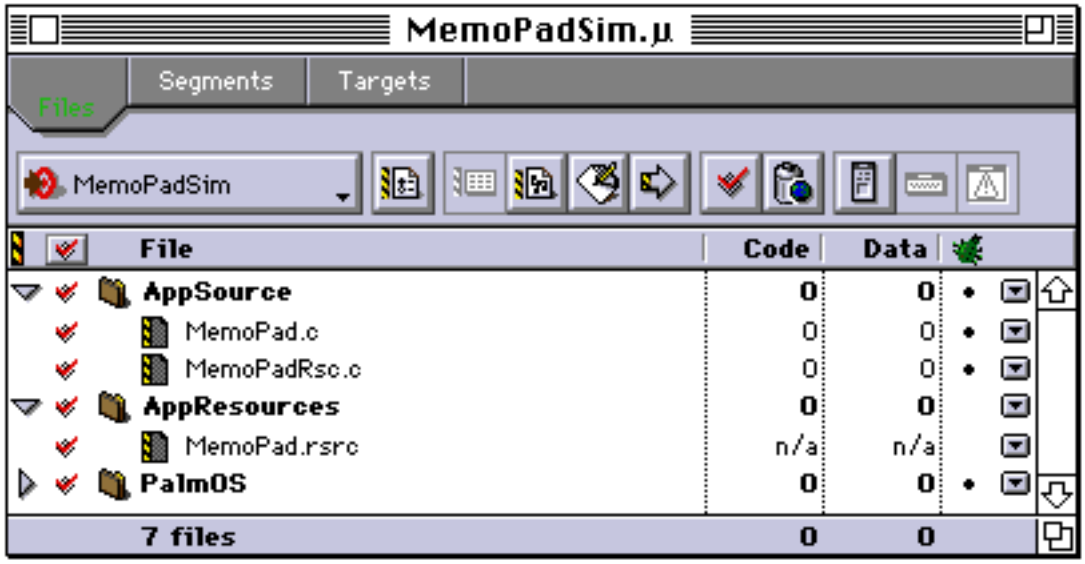
To add resource files to the project, follow these steps:

1. **In the project window, select the placeholder <replace me>.rsrc in the Application Resources area. (Expand Application Resources if needed.)**
2. **Choose Project > Add Files.**
3. **From the dialog box, select the resource file Working Folder\Src\MemoPad.rsrc.**
4. **Click Add.**

Note: You don't need to add header (.h) files like `MemoPadRsc.h` in this process. CodeWarrior is smart enough to find these, based on the project's access paths.

---

5. **Select <replace me>.rsrc, then choose Project > Remove Selected Items to remove the file from the project.**



Setting File Access Paths

If you move your project to a place other than `Apps:Working Folder`, you may have to reset the access paths.

You can do this by choosing the Edit > *ProjName* Settings menu item, then selecting Target > Access Paths in the dialog. See the CodeWarrior documentation for further instructions. For the time being, accept the defaults provided.

With project setup complete, you can either continue by examining the code for this section or go on to "Building and Downloading the Application" on page 37. Understanding the code is a crucial part of this tutorial, so you should come back to "Examining MemoPad.c" on page 32 if you skip it now.

Examining MemoPad.c

To examine `MemoPad.c`:

1. **Double-click on MemoPad.mcp in the Working Folder.**

   This loads Metrowerks CodeWarrior IDE with the project window `MemoPad.mcp` opened.

2. **Double-click MemoPad.c.**

Note: In CodeWarrior, choose Edit > Preferences > Fonts & Tabs, then set Tab Size to 2 for best viewing of any sample code provided with the tutorial.

In the preprocessor area, `MemoPad.c` includes:

- `Pilot.h`, which includes the header files for the Palm OS system functions and data structures that applications usually need.
- `MemoPadRsc.h`, which contains all the application resource defines. This file is created by Constructor, using the resource names that you provide.

Function prototypes for the file follow the macro definitions:

- The StartApplication Function

- The EventLoop Function
- The MainFormHandleEvent Function

The PilotMain Function drives the three other functions.

Note to Windows Users: In the Explorer, select Views > Options. Select "Show All Files" and DO NOT check "Hide MS-DOS file extensions".

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.

Many programs pass parameters to `PilotMain` that can influence its operation. In this simple case, these parameters are ignored.

`PilotMain` usually performs essentially the same way in any application:

- Checks for launch codes. In this example, `PilotMain` only checks for `sysAppLaunchCmdNormalLaunch.`
- Calls `StartApplication.`
- Calls `EventLoop.`

The StartApplication Function

The `StartApplication` function:

- Initializes the required data structures or system facilities.
- 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.

The EventLoop Function

The `EventLoop` function (found just above `PilotMain`) processes events going though these steps:

- `EvtGetEvent` retrieves the next available event from the system event queue 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's not seen or processed by subsequent event handlers.

The `EventLoop` function processes all the events that flow through the application as it executes. `EventLoop` returns when an event is encountered that directs the application to quit. When control returns to `PilotMain` from `EventLoop`, the application quits, returning an error code of zero.

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.

The MainFormHandleEvent Function

If `SysHandleEvent` doesn't handle the event (indicated by a return value of zero), the event is passed to the `MainFormHandleEvent` function.

The `MainFormHandleEvent` is an application-defined function found just above the `EventLoop` routine in the file. It 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 system puts a `ctlSelectEvent` on the event queue when the user has tapped a button. The event structure contains the button's identity.

Currently the Exit button—the only button on the form—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. If no further processing is necessary

for the event, the `MainFormHandleEvent` routine returns `TRUE`. This tells subsequent event handlers not to process the event.

The FrmHandleEvent Function

If `MainFormHandleEvent` doesn't handle the event, it's passed to the `FrmHandleEvent` function for default processing, if any.

Examining MemoPadRsc.h

With CodeWarrior still running, open `MemoPadRsc.h`, which defines names for the application's resources.

Note: Examining the `MemoPadRsc.h` file may be interesting at various phases of the tutorial. However, because CodeWarrior automatically generates the file, it's only rarely discussed in this tutorial. Feel free to investigate it on your own during later phases.

In the Palm OS 2.0 SDK, CodeWarrior creates the `*Rsc.h` file and adds all necessary resources, using the names you assign to them. This file is written each time the `.rsrc` file is saved by Constructor. You don't have to add resource names yourself.

`MemoPadRsc.h` contains definitions for all the resource IDs that constitute the application's interface to the outside world. The `MemoPadRsc.h` file for Phase 1 lists the following resources:

```
#define MemoPadMainForm        1000
#define MemoPadMainExitButton  1001
```

The properties are included as comments.

The macro definitions make it possible to pass a resource ID to a function meaningfully, using the resource's name.

Note: Be sure to assign a unique name to each resource within a given resource type. Duplicate names, for example, two "Options" menus, may result in problems with CodeWarrior. At a minimum, may be confusing for you or people maintaining your code.

Building and Downloading the Application

To build and download an application, follow these steps:

1. **Make sure a device is correctly connected with the desktop computer.**
2. **Open your project (.mcp file) in CodeWarrior.**
3. **Choose Project > Remove Object Code and OK the dialog.**
4. **Choose Project > Reset Project Entry Paths.**
5. **In the Project menu, check that Disable Debugger is displayed. If Enable Debugger is displayed instead, select it to enable the debugger.**

   The debugger should be enabled if you want to debug the application you're building for the device.
6. **Type F7 to build (Make) the executable.**

   CodeWarrior creates a new .prc file and also a new .prc.psym file.
7. **In the Explorer, double-click the newly created .psym file.**

   CodeWarrior prompts you to enter the "shortcut ..2" sequence on the device (That's the shortcut stroke, two taps, and the number 2).
8. **Turn on the device and enter the sequence: ⌕ .. 2, then OK the dialog on the PC screen.**

   Tip: To actually see the text you're entering, tap the Find icon .

   The application is downloaded to the device. A dialog box displays the progress.

   Next, the Debugger launches the program and stops execution at the first line of the `PilotMain` function.
9. **You can now use the CodeWarrior Debugger to debug the application that's on the device. See the CodeWarrior Documentation for more information.**
10. **When you're finished, choose File > Exit to quit the Debugger.**

    This should reset the device. If it doesn't, reset it yourself to conserve batteries.

Debugging Hints

There are two things you should note when debugging with the CodeWarrior Debugger:

- If you start the debugging session with the PalmPilot device in the Memory application, you have to tap the screen once after executing the "shortcut .2" sequence before the application is launched by the Debugger.
- At the end of the debugging session, quit the debugger via the normal means if possible (File > Exit). This allows the debugger to clean up on the device and reset the device. If you don't quit normally, two potential problems may result:
  - The device's serial port remains open. This uses up batteries and prevents HotSync from working.
  - Any breakpoint code that the debugger inserted into the target application will still be there. This can only be cleaned up by exiting the debugger gracefully (File > Exit) or deleting the application.

Adding a Menu and a Simple Dialog

Overview

In Phase 2 you build a simple menu. Users can select this Get Info menu item to bring up an Info dialog.

This phase explains how to do this in the following sections:

Setup

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

Note: If you've just completed Phase 1, you don't need to copy `MemoPad.mcp` or the two `MemoPad.rsrc` files.

Copy these files...	From...	To...
`MemoPad.c`	`Tutorial\MemoPad 02\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 01\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 01\Src\Resource.frk`	`Apps\Working Folder\Src\Resource.frk`
`MemoPad.mcp`	`Tutorial\MemoPad 01`	`Apps\Working Folder`

Adding an Options Menu With a Get Info Item

Background

A menu user interface usually consists of these pieces:

- 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 or commands (New, Open, Close, Save, Quit, and so on).

Palm OS follows this same hierarchy, and you can quickly implement it using Constructor.

Creating the Menu Bar

To create the menu bar, follow these steps:

1. **From Constructor, choose File > Open Project File and select Src\MemoPad.rsrc.**

   Tip: You can also double-click on `MemoPad.rsrc` in the IDE project window.

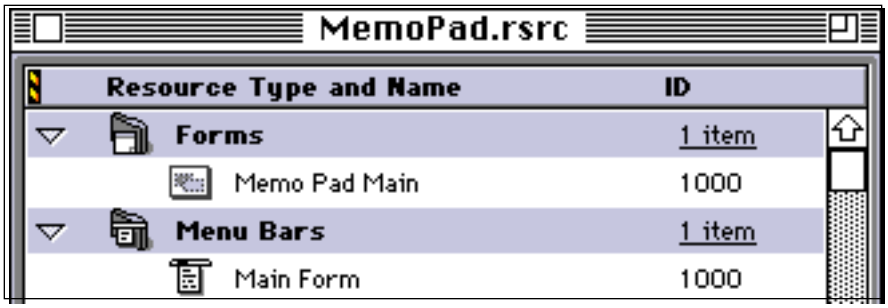
   Constructor displays the Constructor project window.

2. **In the Constructor project window, select Menu Bars and type Ctrl-K.**

   Constructor creates a menu bar instance and displays the associated icon immediately below Menu Bars.

3. **Rename the "untitled" menu bar to "Main Form".**

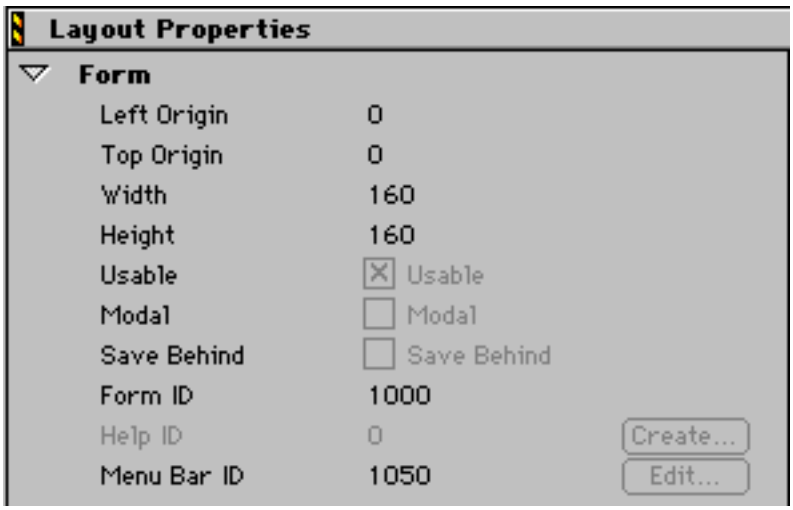
   You can change the name in the Constructor project window. The default ID is assigned by Constructor (but it can be changed by the developer).



4. **If the list of forms isn't already visible, double-click on Forms.**
5. **Double-click on Memo Pad Main.**

   The form with the Exit button from the last phase is displayed.
6. **In the Layout Properties panel of the form, enter the ID of the newly created menu bar into the Menu Bar ID field.**

   This assigns the menu to that form. (Note that the ID number may be different from the one in the illustration below.)



Creating a Menu

To add a menu with one menu item to the menu bar, follow these steps:

1. **In the Constructor project window, select the Menus icon, then type Ctrl-K to create a new menu.**
2. **Change the menu name to "Main Options".**

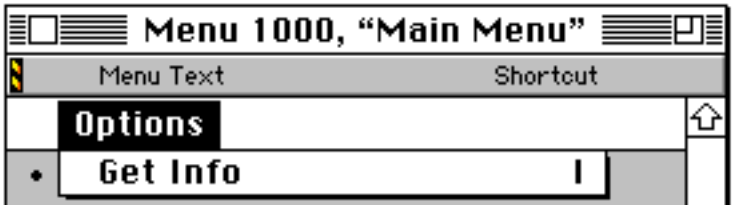
   This name is appropriate because you'll later create an Edit Options menu.
3. **Double-click the Main Options icon.**

   Constructor displays an "untitled" menu in the Menu Editor.
4. **Change the menu from "untitled" to "Options".**

   Next, you add a menu item to the menu:
5. **Type Ctrl-K (or choose Edit > New Menu Item).**

   A new "untitled item" appears.
6. **Change the item to "Get Info".**
7. **With the cursor still in the menu, press the Tab key, then "I".**

   The Shortcut key is separated from the menu name by a Tab key.



Note: There's no need to add the stroke character before the letter "I." The system does that for you when it displays the menu.

8. **Close the Menu Editor window.**

Adding the Menu to the Menu Bar

To add the menu to the menu bar, follow these steps:

1. **Double-click the menu bar instance "Main Form".**

   The Menu Bar Editor appears.

2. **Drag the Main Options menu you just created from the Constructor project window onto the Menu Bar Editor.**

   Constructor adds the menu to the menu bar.

3. **Close the Menu Bar Editor and save the resource file.**

A Note on Menu Naming

Constructor generates the constants for menu items (commands) based on the menu's Menu ID, which will be the same as the Resource ID for new menus. Constructor doesn't allow you to edit the menu item's ID. The first menu item will have the same number as the menu's Menu ID. Subsequent menu items have resource IDs 1 greater than the previous menu item.

For example, assume you have an Edit menu with a Resource ID of 1000. This results in the following numbering:

Menu Item	
menu item 1 (Undo)	menu Resource ID (1000)
menu item 2 (Cut)	menu Resource ID + 1 (1001)
menu item 3 (Paste)	menu Resource ID + 2 (1002)

Earlier versions of this SDK used ResEdit, which allowed developers to edit the menu item ID independently of the resource ID of that menu. In Constructor, on the other hand, modifying the Resource ID automatically changes the menu ID to match it.

To avoid collisions of resource IDs for menu items, you should spread the resource IDs of your menus by an appropriate number; at least 10, that is, 1000, 1010, 1020, and so on.

Code Changes: Simple Menu Behavior

In this section, you learn how `MemoPad.c` has been modified to respond correctly when the user chooses the new menu item that was added to the GUI.

This section discusses only those new pieces that are relevant to the UI elements you defined so far. Some other changes are discussed further below. These changes were made to `MemoPad.c` to handle UI elements that will be added later in this chapter.

The file `MemoPad.c` in the `Src` folder includes code changes to incorporate a menu bar with an Options menu.

Note: You can find the comments for a phase by searching for the string `Px`, that is, search for P2 to find Phase 2 comments.

You learn about these topics:

Changes to MemoPadRsc.h

The `MemoPadRsc.h` file now contains `#define` macros for the menu bar, the Options menu, and the Get Info command.

The New Menu Event Handler in MemoPad.c

To incorporate a menu bar with an Options menu, the following changes were made:

- `CurrentMenu` is a new global variable that keeps track of the currently active menu by way of a pointer.

  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:

  - First checks for the need to dispose of an existing menu. (Note that `CurrentMenu` was initialized to `NULL`.)
  - Then initializes the new menu with a call to `MenuInit`.
- `MainFormHandleEvent` now contains a test for a `menuEvent`. The identity of the menu item is included in the event structure. In this phase, there is only one menu item, so the program knows it's the Get Info item.

  Choosing the menu item triggers a series of actions, culminating in the appearance of the Info dialog. These actions are described in detail under A New Handler for the Info Dialog.
- The `EventLoop` routine now includes 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.

Adding a Get Info Form With an OK Button

With a menu structure in place, you now need to add a dialog that appears when the user chooses the menu command, and an OK button to return to the Main form. In this section, you'll build the dialog; a new form with the following resources:

- 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 resource with an associated bitmap for the company logo.

Creating a New Form and Its Components

To create the new form, the OK button, and two labels to display information follow these steps:

1. **Return to MemoPad.rsrc in Constructor.**
2. **In the Constructor project window, select Forms and type Ctrl-K.**

   A new form instance is displayed below Forms in the Constructor project window.
3. **Name the form "Memo Pad Info", then open it.**

   The form is displayed in a Form Editor.

   Constructor creates a new Form Editor for each form.
4. **Set the properties of the form as follows:**

Left and Top Origin	2, 46
Width and Height	156, 112
Usable	Checked
Modal	Checked
Save Behind	Checked
Form ID	(Assigned by Constructor)
Form Title	Info

Note that Modal and Save Behind are checked. This is appropriate for a dialog and distinguishes it from full-screen forms.

5. **In the Catalog window, select the Button icon and drag it onto the new Form.**
6. **The Button should have the following properties:**

Object Identifier	OK
Button ID	(Assigned by Code Warrior)
Left and Top Origin	60, 95

Width and Height	36, 12
All check boxes	Leave checked
Font	Choose Standard from the popup menu
Label	OK

7. **Add a Label to the Form.**
8. **The Label should have the following properties:**

Object Identifier	Appl Name
Label ID	(Assigned by Code Warrior)
Left and Top Origin	50, 28
Usable	Leave checked,
Font	Choose **Bold** from the popup menu.
Text	Memo Pad

9. **Add a second Label with the following properties:**

Object Identifier	Version
Label ID	(assigned by Code Warrior)
Left and Top Origin	50, 39
Usable	Leave checked.
Font	Bold
Text	Version 2.0

After you've made all the changes, the form should look as follows:

Adding Help Text to the Form

In this section, you'll add help text to the form.

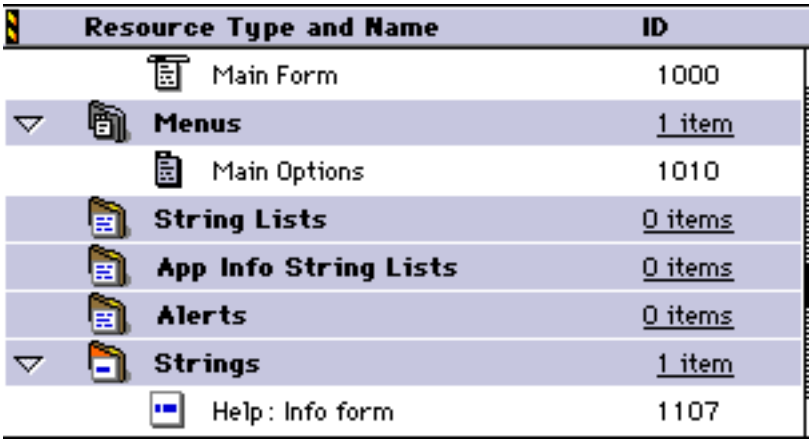
Help text is a standard feature of Palm OS dialog boxes. If there is help text, the letter "i" (for "information") appears in the top-right corner of the dialog in the title bar. The text is displayed when the user taps the "i".

To add help text, follow these steps:

1. **In the Constructor project window, select Strings and type Ctrl-K.**

   A new "untitled" string appears.

2. **Change the name of the string resource from "untitled" to "Help: Info form".**



3. **Double-click the icon.**

   The String Editor appears. You can type all text for a String resource into the String Editor.

4. **Type into the String Editor:**

   ```
   The info dialog provides information about the
   application such as the development company and
   version number.
   ```

5. **Close the String Editor.**

6. **In the Forms window, click anywhere on the Memo Pad Info form to select it.**

   You can now edit the Layout Properties of the form.

7. **In the Help ID field, enter the ID of the String resource you just created.**

   Constructor adds the "i" button to the top-right corner of the form. When the user taps this button on the device, the system displays the help text.

Tip: You can also assign a Help string to a dialog by assigning a Help ID to the form in the Form Editor, then choosing Create.

Adding a Bitmap to the Info Form

When an application wants to display a logo or other graphic information in a form, it needs to add a Form Bitmap resource. In this example, you'll actually resize the editor and paste an existing bitmap into it.

To create and add a bitmap, follow these steps:

1. **In the Catalog window, select the Form Bitmap icon and drag it onto the Info form.**

2. **The Form Bitmap should have the following properties:**

Object ID	(Assigned by Code Warrior)
Object Identifier	Company Logo

Left and Top Origin	5, 25
Bitmap Resource ID	1110 (Assigned by developer)
Usable	(leave checked)

3. **Click the Create button next to the Bitmap Resource ID.**

   The Bitmap Editor appears.

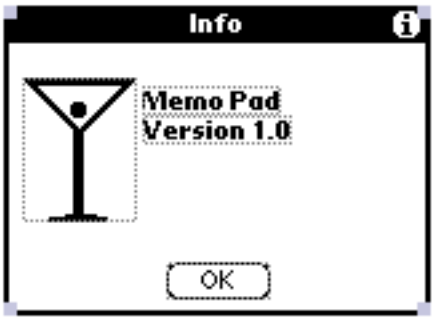
4. **Choose "Options > Set image size" and set the image size to:**

   - width: 43
   - height: 54

5. **Use the Bitmap Editor to create a bitmap of your choice.**

6. **Close the Bitmap Editor.**

   The bitmap you created becomes visible on the Info form. Below is one example, your form will differ depending on the bitmap you created.



7. **Save the resource file.**

Code Changes: Info Dialog

In this section, you learn about additions that were made to the source code to achieve the desired behavior.

Changes to MemoPadRsc.h

Constructor generates a `#define` name for every resource in the application. For example, `MemoPadRsc.h` now has a `#define` for the `infoForm` resource ID. The `infoForm` macro has the same value as its ID.

Applications don't usually have to refer to all of the components of a form explicitly because the Palm OS UI manager handles most of the default resource interaction.

A New Handler for the Info Dialog

The file `MemoPad.c` includes code changes required to use the Info dialog box. When the user taps the "i" (info) button, the system displays the Info form. When the user taps the OK button on the Info form, the display is restored.

Open `MemoPad.c` from CodeWarrior and take a look at the changes.

The `MainFormHandleEvent` routine now contains a test for a `menuEvent`. 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.

Building and Downloading the Application

To build and download an application for use with the CodeWarrior Debugger, follow these steps:

1. **Delete any previous MemoPad applications on the device and make sure it's correctly connected with your desktop computer.**

   See Deleting the MemoPad Application for more information.

2. **Open your project (.mcp file) in CodeWarrior.**

3. **Choose Project > Remove Object Code and OK the dialog.**

4. **Choose Project > Reset Project Entry Paths.**

5. **In the Project menu, check that Disable Debugger is displayed. If Enable Debugger is displayed instead, select it to enable the debugger.**

   The debugger should be enabled if you want to debug the application you're building for the device.

6. **Type F7 to build (Make) the executable.**

   CodeWarrior creates a new .prc file and also a new .prc.psym file.

7. **Double-click the newly created .psym file.**

   CodeWarrior prompts you to enter the "shortcut .2" sequence on the device.

8. **Turn on the device and enter the sequence: ♌ .. 2, then OK the dialog on the PC screen.**

   The application is downloaded to the device. A dialog box displays the progress.

   Next, the Debugger launches the program and stops execution at the first line of the `PilotMain` function.

9. **You could now debug the application on the device just as you would debug a resident application.**

   See the CodeWarrior documentation for more information.

Preparing the Device for Testing

1. **To exercise the application in standalone mode, choose File > Exit to quit the Debugger.**

   This should reset the Device.

2. **If exiting the Debugger hasn't reset the device, reset it by hand.**

   See Resetting the Device for more information.

3. **Choose the Applications silk-screened button and select the MemoPad (not Memo Pad) application in the launcher.**

Exercising the Application

To try out the menu system, follow these steps:

1. **On the device, tap the silk-screen Menu button.**

   The Options menu and the Get Info menu item are displayed.

2. **Select the Get Info item.**

   The Info form is displayed.

3. **Select the Info (i) icon in the top-right corner of the form's title bar.**

   The Help text you created for the Info form is displayed.

4. **Select the Done button.**

   This dismisses the Help form.

5. **Select the OK button.**

   The Info form disappears.

6. **Select the Main form's Exit button.**

   This exits the application.

Adding an Edit Form and Navigation Between Forms

Overview

Phase 3 adds a New button, a full-screen Edit form, and some navigation between the Edit form and the Main form.

It discusses these topics:

Setup

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

Note: If you've just completed Phase 2, you don't need to copy `MemoPad.mcp` or the two `MemoPad.rsrc` files.

Copy these files...	From...	To...
`MemoPad.c`	`Tutorial\MemoPad 03\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 02\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 02\Src\Resource.frk`	`Apps\Working Folder\Src\Resource.frk`
`MemoPad.mcp`	`Tutorial\MemoPad 02`	`Apps\Working Folder`

Adding a New Button to the Main Form

This section shows you how to create a New button for the Main form. This button, when tapped, displays the Edit form.

To create a New button for the Main form, follow these steps:

1. **Double-click MemoPad.rsrc to launch Constructor.**

   If Constructor is already open, choose File > Open Project File and select `MemoPad.rsrc`.

2. **Double-click Memo Pad Main to display the Form Editor.**
3. **In the Catalog window, select a button and drag it onto the form.**

   Constructor creates a Button resource.

4. **Assign the following properties to the button:**

Object Identifier	New
Button ID	(Assigned by CodeWarrior)
Left and Top Origin	60, 146
Width and Height	40, 12
All check boxes	Leave checked
Font	Standard
Label	New

5. **After you've added the button, save the resource file.**

Code Changes: Event Handler for New Button

The file `MemoPad.c` includes the following changes, which were required to use the New button.

- A new global variable, `CurrentView`, keeps track of the current form.
- The `MainFormHandleEvent` routine no longer contains a simple `if` statement but 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.
- 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.

Creating an Edit Form

This section steps you through creating an Edit form with a Title and a Done button.

The first version of the Edit form is similar to the first version of the Main form. It contains only a title and an Exit button. When the user taps the Exit button, the Main form is displayed again.

Creating the Form

If you remember how to create a form from the last phase, create a new form, name it "Memo Pad Edit", and give it the following properties:

Left and Top Origin	0, 0
Width and Height	160, 160
Usable	Checked
Modal, Save Behind	Unchecked
Form ID	(Assigned by Constructor)
Form Title	Edit Memo

Otherwise, follow these steps:

1. **In the Constructor project window, select Forms and type Ctrl-K.**

   Constructor displays a new Form below the already existing Main form.

2. **Name the new Form Memo Pad Edit, then double-click on it.**

   Constructor displays the Form in a new Form Editor.

3. **Set the properties to the ones in the table above.**

   Most likely, you only have to change the Form Title.

Adding a Done Button

The Done button in the Edit form won't be centered because other buttons will be added later. To add a Done button to the Edit form, follow these steps:

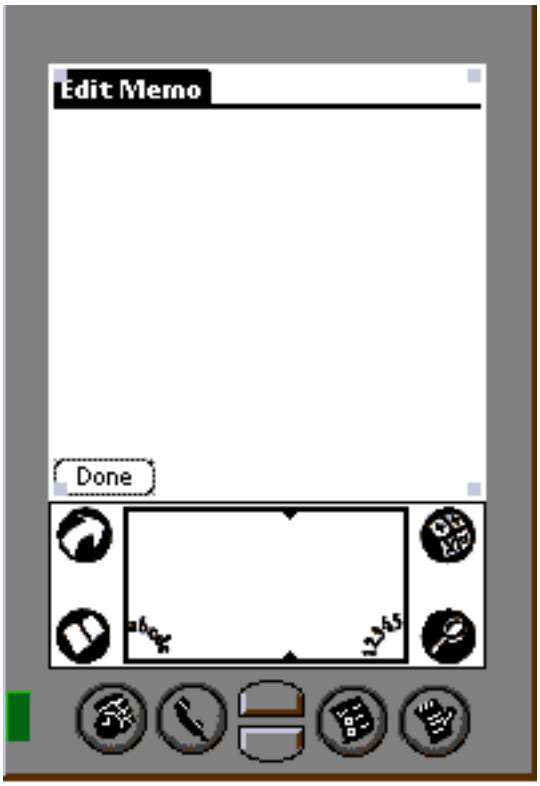
1. **In the Catalog window, select a button and drag it onto the Form.**

   Constructor creates a Button resource.

2. **Assign the following properties to the button:**

Object Identifier	Done
Button ID	(Assigned by Constructor)
Left and Top Origin	1, 147
Width and Height	36, 12
All check boxes	Leave checked
Font	Standard
Label	Done

3. **After you've added the button, save the resource file.**


Edit Memo
Done

Code Changes: Edit Form Handling

This section discusses changes required to accommodate the new Edit form. You learn about Adding Edit Form Handling to the Code and Other Changes to the Application Code.

Adding Edit Form Handling to the Code

The file `MemoPad.c` has been changed so that it's possible 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: tapping the Done button and opening a form. This example has two events to be handled, `ctlSelectEvent` and `frmOpenEvent.`
- If the event type is `ctlSelectEvent`, the user has tapped a button. The button ID is included in the event structure. The function checks whether the ID matches the ID of the Done button. If it does, it closes the Edit form and redisplays the Main form using a call to `FrmGotoForm.`

  `FrmGotoForm` redisplays the Main form by adding `frmCloseEvent`, `frmLoadEvent` and `frmOpenEvent` events to the event queue. The application retrieves these later in the `EventLoop` routine.
- If the event is `frmOpenEvent`, the user has tapped the Main form's New button. 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.

Other Changes to the Application Code

Several other changes have been made to the example code for this phase.

- In the `PilotMain` routine, a call to `FrmGotoForm` with `CurrentView` as an argument follows `StartApplication`.

  The 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.

- In the `EventLoop` routine, the calls to `ApplicationHandleEvent` and `FrmDispatchEvent` replace calls to `MainFormHandleEvent` and `FrmHandleEvent`.

- 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.

- `ApplicationHandleEvent` now displays the correct form—Edit form or Main form—when passed an event.

  `ApplicationHandleEvent` 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.

- The resource for the Main form was modified to include setting the menu resource ID to the ID of the menu bar resource. As a result, the default form-event processing routine, `FrmHandleEvent` (called internally by `FrmDispatchEvent`) can handle setup and teardown of the menu for a form. As a result, `SetCurrentMenu` is no longer required, because `FrmSetActiveForm` handles this task automatically.

Building and Downloading the Application

To build and download an application for use with the CodeWarrior Debugger, follow these steps:

1. **Delete any previous MemoPad applications on the device and make sure it's correctly connected with your desktop computer.**

   See Deleting the MemoPad Application for more information.

2. **Make sure your device is correctly connected with your desktop computer.**

3. **Open your project (.mcp file) in CodeWarrior.**

4. **Choose Project > Remove Object Code and OK the dialog.**

5. **Choose Project > Reset Project Entry Paths.**

6. **In the Project menu, check that Disable Debugger is displayed. If Enable Debugger is displayed instead, select it to enable the debugger.**

   The debugger should be enabled if you want to debug the application you're building for the device.

7. **Type F7 to build (Make) the executable.**

   CodeWarrior creates a new .prc file and also a new .prc.psym file.

8. **Double-click the newly created .psym file.**

   CodeWarrior prompts you to enter the "shortcut .2" sequence on the device.

9. **Turn on the device and enter the sequence: ♌ .. 2, then OK the dialog on the PC screen.**

   The application is downloaded to the device. A dialog box displays the progress.

   Next, the Debugger launches the program and stops execution at the first line of the `PilotMain` function.

10. **You could now debug the application on the device just as you would debug a resident application.**

    See the CodeWarrior documentation for more information.

Preparing the Device for Testing

1. **To exercise the application in standalone mode, choose File > Exit to quit the Debugger.**

   This should reset the Device.

2. **Remove the device from the cradle.**

3. **If exiting the Debugger hasn't reset the device, reset it by hand.**

   See Resetting the Device for more information.

4. **Choose the Applications silk-screened button and select the MemoPad (not Memo Pad) application in the launcher.**

Exercising the Application

To exercise the new features of MemoPad Phase 3, follow these steps:

1. **Select the Main form's New button.**

   The program displays the Edit form.

2. **Select the Menu button.**

   Notice that there is no menu for this form.

3. **Select the Done button in the Edit form.**

   The program returns to the Main form.

4. **Select the Exit button on the Memo Pad Main form to quit the application.**

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. You'll learn how to set up basic edit commands and field navigation commands.

The phase discusses the following topics:

Setup

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

Note: If you've just completed Phase 3, you don't need to copy `MemoPad.mcp` or the two `MemoPad.rsrc` files.

Copy these files...	From...	To...
`MemoPad.c`	`Tutorial\MemoPad 04\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 03\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 03\Src\Resource.frk`	`Apps\Working Folder\Src\Resource.frk`
`MemoPad.mcp`	`Tutorial\MemoPad 03`	`Apps\Working Folder`

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 add a UI object that displays the Graffiti Shift symbol (as well as other extended character symbols) above the Graffiti writing area.

Follow these steps to add a text field to the Edit form:

1. **In Constructor, double click on Forms if all Form instances aren't visible, then double-click on "Memo Pad Edit".**

   Constructor displays the Form Editor.

2. **In the Catalog window, select the Field icon and drag it onto the Layout Appearance panel, dropping it onto the form.**

   You've created a field that's now part of the form.

3. **Give the field the following properties:**

Object Identifier	Edit Field
Field ID	(Assigned by Constructor)

Left and Top Origin	10, 17
Width and Height	140, 125
Usable, Editable, Underline, Left Justified	Leave checked
Single Line, Dynamic Size, Auto Shift, Has Scroll Bar, Numeric	Leave unchecked
Max chars	1028
Font	Standard

Note: The object attributes Auto Shift, Has Scroll Bar, etc., are left in their default settings because the code for this phase doesn't support that behavior. In later phases, you'll work with fields that have scrollbars and have auto-shift set. For more information, see also the "Code Warrior Constructor for Palm OS Manual."

Adding a Graffiti Shift Indicator

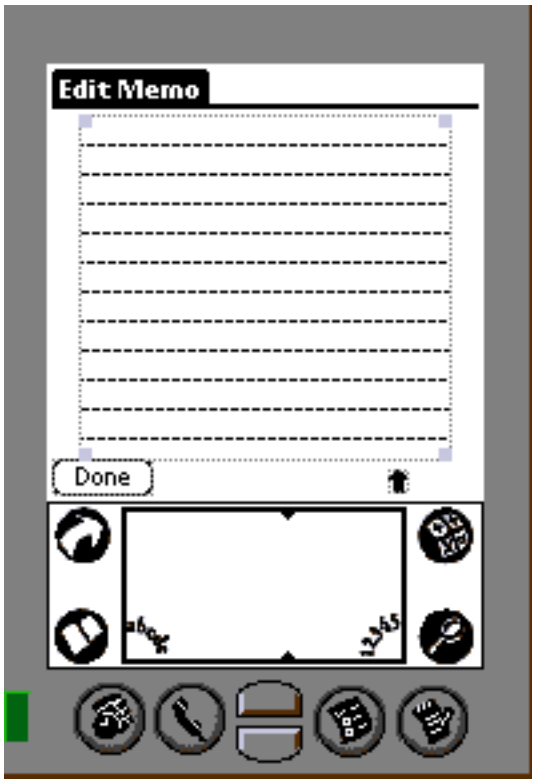
The Graffiti Shift Indicator specifies where the Graffiti symbols (for punctuation, symbol, uppercase shift, and uppercase lock) appear on the form.

To add a Graffiti Shift Indicator, follow these steps:

1. **In the Catalog window, select the Graffiti Shift Indicator icon and drag it onto the Layout Appearance panel, dropping it onto the form.**

   Constructor creates a Graffiti Shift Indicator.

2. **Set the Object Identifier to "Graffiti Shift".**
3. **To place the resource in the lower right of the display, set the Left Origin to 126 and Top Origin to 150.**
4. **Save the resource file.**



Code Changes: Revised 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. The changes support the new resources in Phase 4.

Within the `EditFormHandleEvent` routine, there is now a `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.

Adding Two Menus for the Edit Form

This section illustrates:

- Adding Menus and Menu Items. This includes creating the menu, adding menu commands, and adding shortcuts.
- Joining Menus, Menu Bar, and Edit Form

Adding Menus and Menu Items

The process of adding an Edit menu resembles that for the Options menu in Phase 2. As before, the process consists of three parts:

- Creating a menu bar.
- Creating menus and menu items and dragging them onto the menu bar.
- Associating the Menu Bar with the form.

The remainder of this section steps you through these actions.

Creating the Menu Bar

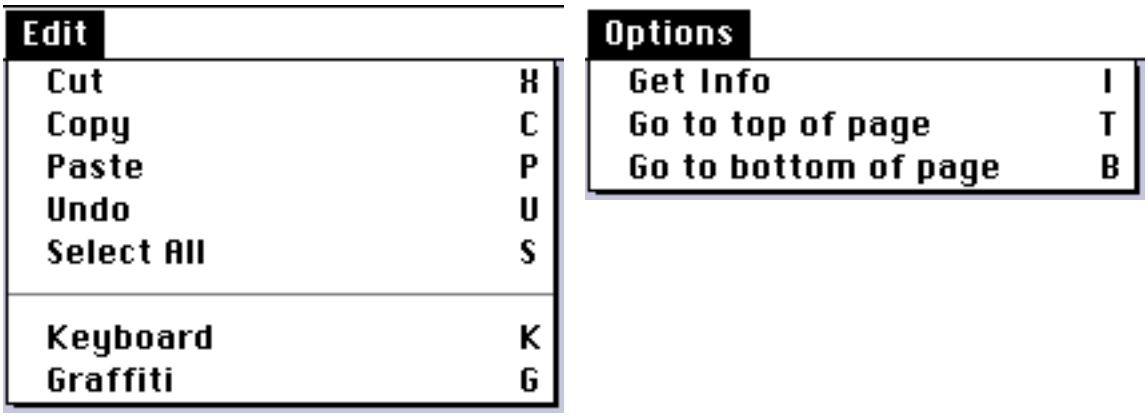
If you already know how to create a Menu Bar, do so and name it Edit Form Menu. To learn an alternative process, follow these steps:

1. **In the Constructor project window, open Menu Bars to display all instances.**
2. **Select Menu Bars and type Ctrl-K.**

   Constructor creates an new menu bar.
3. **Change the name of the new Menu Bar to “Edit Form”.**

Creating the Menus

The next step is to create two menus and add menu items in the Menu Editor for each. The menus will look like the following picture:



If you remember how to create menus from the previous phases, feel free to do so. The two menus are called "Edit Edit" and "Edit Options".

Otherwise, follow these steps:

1. **Double-click on the "Edit Form" Menu Bar.**

   Constructor displays the Menu Editor.

2. **In the Constructor project window, select menus and type Ctrl-K twice to create two menus.**
3. **Name one menu "Edit Options" and another "Edit Edit".**
4. **Double-click on "Edit Edit".**

   An "untitled" menu appears in the Menu Editor.

5. **Change the menu text to "Edit."**
6. **With Edit selected, type Ctrl-K 5 times to add the menu items; name them as follows:**

Tip: You can add the Shortcut by typing a Tab after the menu item, then the single letter. The stroke character is later added by the program.

Item	Shortcut
Cut	X
Copy	C
Paste	P
Undo	U
Select All	S

7. **Type Ctrl- (minus), or choose Edit > New Separator Item.**
8. **Type Ctrl-K twice more to add these menu items:**

Item	Shortcut	Functionality
Keyboard	K	Displays on-screen keyboard to allow users to enter text.
Graffiti	G	Displays Graffiti Reference screen.

9. **Close the Edit Menu Editor, then double-click on Edit Options and repeat the process to create the Options menu:**

Item	Shortcut
Get Info	I
Go to top of page	T
Go to bottom of page	B

10. **Close the Options Menu Editor.**

Joining Menus, Menu Bar, and Edit Form

With the menus completed, you can add them to the menu bar, then the menu bar to the Edit form. To do this, and verify the IDs, follow these steps:

1. **Double-click the menu bar "Edit Form".**

   The Menu Bar Editor appears.

2. **Drag first the "Edit Edit" menu, then the "Edit Options" menu onto the Menu Bar Editor, then close the Menu Bar Editor.**

   The menus are now attached to the menu bar.

3. **In the Form Editor for the Edit form, select the Form in the Layout Appearance panel (be sure to click outside the field).**

4. **In the Layout Properties panel, enter the Menu Bar ID, the number generated by Constructor for the "Edit Form" menu bar into the Menu Bar ID field.**

5. **Save the resource file.**

Code Changes: Edit Menu

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

Changes to MemoPadRsc.h

To identify the resources to the application code, there are now a `#define` macro for the Edit menu and the new menu command resource IDs in `MemoPadRsc.h`, which is generated by Constructor.

An Event Handler for the Edit Menu

The file `MemoPad.c` includes changes required to make the new menus in the Edit form usable.

In the `EditFormHandleEvent` routine, there is now a new case for `menuEvent`.

- First, `MenuEraseStatus` clears the menu command prompt from the display.
- Then, `EditDoMenuCommand` executes the menu command, based on the command ID that was extracted from the event structure.

A Menu Item Handler for the Edit Form Menu

`EditDoMenuCommand` is a new routine located just above `EditFormHandleEvent`. It's responsible for handling all the menu commands for the Edit form.

- **getInfo—**The `getInfo` command displays the Info form using the same actions as were put in place earlier for the `menuEvent` in the `MainFormHandleEvent` routine.
- **goToTop—**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 causes the text to scroll in the field, when necessary.

- **goToBottom—**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 causes the text to scroll in the field, when necessary.
- **cutCmd—**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.
- **copyCmd—**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.
- **pasteCmd—**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.
- **undoCmd—**The undoCmd command reverses the last text change. FldUndo handles this including the necessary display updates.
- **selectAllCmd—**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.
- **keyboardCmd—**The keyboardCmd command displays the onscreen keyboard. SysKeyboardDialog handles display of the keyboard and all its interactions, 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.
- **graffitiRefCmd**—The graffitiRefCmd command displays the Graffiti reference screen.

Adding a ROM-Incompatible Alert

Because this version of the tutorial uses a function that's only available in version 2.0, you need to add code to check the system version (explained in the next section) and an alert that the code can pop up when the requirement hasn't been met.

To add the alert, follow these steps:

1. **In the MemoPad Project window in Constructor, select Alerts and type Ctrl-K.**

   Constructor creates an Alert instance.

2. **Name the instance "Rom Incompatible".**

3. **Double-click the newly-created alert.**

   The Alert Editor appears.

4. **Assign the following properties to the alert:**

Alert Type	Error
Title	System Incompatible
Message	System Version 2.0 or greater is required to run this application.
Item Text 0	OK

5. **Save the resource file.**

Code Changes: System Version Check

The `graffitiRefCmd` function, which displays the Graffiti Reference screen, is only available in Palm OS 2.0. Because of this, Phase 04 contains the necessary code for checking version number and a custom alert to display if the version is too low.

Here are the different components:

- `PilotMain` now checks whether the application runs on a 2.0 device. This is necessary because the function `graffitiRefCmd` is new in Palm OS 2.0. `PilotMain` calls the `RomVersionCompatible` function.
- The `RomVersionCompatible` function determines who requested that the application launch by checking launch codes.
  - If the user selected the application from the launcher, `RomVersionCompatible` displays an alert if the ROM version of the device is too low.
  - For all other launch codes, no alert is displayed.
- To make the custom alert work, there's a `RomIncompatibleAlert #define` in `MemoPadRsc.h`.

Building and Downloading the Application

To build and download an application for use with the CodeWarrior Debugger, follow these steps:

1. **Delete any previous MemoPad applications on the device and make sure it's correctly connected with your desktop computer.**

   See Deleting the MemoPad Application for more information.

2. **Open your project (.mcp file) in CodeWarrior.**
3. **Choose Project > Remove Object Code and OK the dialog.**
4. **Choose Project > Reset Project Entry Paths.**
5. **In the Project menu, check that Disable Debugger is displayed. If Enable Debugger is displayed instead, select it to enable the debugger.**

   The debugger should be enabled if you want to debug the application you're building for the device.

6. **Type F7 to build (Make) the executable.**

   CodeWarrior creates a new .prc file and also a new .prc.psym file.

7. **Double-click the newly created .psym file.**

   CodeWarrior prompts you to enter the "shortcut .2" sequence on the device.

8. **Turn on the device and enter the sequence: ♌ .. 2, then OK the dialog on the PC screen.**

   The application is downloaded to the device. A dialog box displays the progress.

   Next, the Debugger launches the program and stops execution at the first line of the `PilotMain` function.

9. **You can now debug the application on the device just as you would debug a resident application.**

Preparing the Device for Testing

1. **To exercise the application in standalone mode, choose File > Exit to quit the Debugger.**

   This should reset the Device.

2. **Remove the device from the cradle.**

3. **If exiting the Debugger hasn't reset the device, reset it by hand.**

   See Resetting the Device for more information.

4. **Choose the Applications silk-screened button and select the MemoPad (not Memo Pad) application in the launcher.**

Exercising the Application

To try the main additions to this phase, follow these steps:

1. **Select the Main form's New button.**

   The application displays the Edit form. Note the flashing insertion point at the beginning of the text field.

2. **Select the Menu button.**

   The application displays a menu bar with the Options and Edit menus.

3. **Select the Get Info item from the Options menu.**

   The application displays the Info form.

4. **Tap OK to close the Info form.**

5. **Type or write some words into the text field with Graffiti, like "To-to, I think we're not in Kansas any more!".**

   With some text in place you can experiment with moving the insertion point. Follow these steps:

6. **Click on the displayed text to change the location of the insertion.**

7. **Select the Menu button. From the Options menu, choose "Go to top of page".**

   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.

Trying out Edit Commands

To try the Edit commands, follow these steps:

1. **Add some text to the Edit form.**
2. **Click and drag over some text to select it.**
3. **Select the Menu button and choose Edit > Copy.**

   The selected text is copied to the clipboard.

> **Tip:** If you'd like to try the command stroke shortcut, you can enter the command stroke in the Graffiti area.

4. **Choose Edit > Paste from the Edit menu.**

   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. **Choose Edit > Select all to select all the text in the field.**
6. **Choose Edit > Cut item to copy selected text to the clipboard and remove it from the displayed text.**
7. **Choose Edit > Undo to undo the Cut operation.**

Exercising the On-Screen Keyboard

To exercise the on-screen keyboard; follow these steps:

1. **Tap somewhere in an editable region.**

   You have to have an active insertion point to use the keyboard.

2. **Choose the Edit > Keyboard menu command.**

   The application displays the on-screen keyboard.

3. **Use the on-screen keyboard to add some characters to the text.**

4. **To dismiss the keyboard, tap the keyboard's Done button.**

Handling Large Amounts of Text

To see 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. **Choose the Done button to return to the Main form.**

3. **Choose the Main form's Exit button to quit the application.**

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, the application will be able to store and retrieve text data the user enters into the text field of the Edit form.

To allow text storage and retrieval, the code in this phase creates a Palm OS database and within it, a record to store the data.

The phase discusses the following topics:

Setup

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

Note: If you've just completed Phase 4, you don't need to copy `MemoPad.mcp` or the two `MemoPad.rsrc` files.

Copy these files...	From...	To...
`MemoPad.c`	`Tutorial\MemoPad 05\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 04\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 04\Src\Resource.frk`	`Apps\Working Folder\Src\Resource.frk`
`MemoPad.mcp`	`Tutorial\MemoPad 04`	`Apps\Working Folder`

Code Changes: Text Storage in Database Record

Once an application moves beyond simple user interaction and requires text storage, it needs a database. This section explores the basics of Palm OS databases and describes three key functions for running a database. It discusses these topics:

- Database Basics
- Database Create, Open, and Close Functions

Palm OS databases are discussed in detail in "Developing Palm OS Applications, Part II."

Database Basics

Normally, each Palm OS application has its own database for storing its records. The database resides in the Palm OS device unit's storage memory.

Each database has a name, a creator, and a type:

- The name can be up to 31 characters in length, not including the NULL terminator. Because it's the primary identifier, it must be unique within the system.

- The creator is a four-character value (not including the NULL terminator) 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(not including the NULL terminator) that must be either all uppercase or mixed case.

Database Create, Open, and Close Functions

In `MemoPad.c` for Phase 5, you can find the changes that were made to handle the application database:

The two functions in Phase 5 that handle starting and stopping an application both use a database:

- **StartApplication—**`StartApplication` has been revised to create and open a database for this application:

  `StartApplication` looks for an existing application database. The `DmOpenDatabaseByTypeCreator` function returns an ID for the database that matches the requested name.

  – If the database doesn't exist, a call to `DmCreateDatabase` creates it. If database creation fails, there is a call to `ErrFatalDisplay`, and the application exits.

  – Normally, database creation succeeds, so `DmOpenDatabaseByTypeCreator` 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 `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.

  The application's database is kept open while the application is running.

- **StopApplication**—`StopApplication` is a new routine that calls `DmCloseDatabase`. `DmCloseDatabase` cleans up the state of the application's data and closes the database. `StopApplication` is called by `PilotMain` just before exiting the application.

Code Changes: Revised Handler for Done Button

Addition of the Edit form required changes in several places. These changes ensure that the text the user enters is saved to the database created (or opened) in the application start routine.

This section first discusses changes to EditFormHandleEvent, then the new function EditSaveData.

EditFormHandleEvent

`EditFormHandleEvent` now saves the Edit form's text-field data to the application's database when the user taps Done.

If the `CtlSelectEvent` case finds that the Done button has been selected, it does the following:

- Gets a pointer to the edit field's data structure.
- Passes the pointer to a new routine named `EditSaveData` (see next section).
- Resets the `CurrentRecord` global variable to indicate that there is no longer a current record.

EditSaveData

The `EditSaveData` function writes text field data to the first record of the application's database. Here's what `EditSaveData` does:

- Gets a pointer to the text field and tests the pointer for data in the field to ensure that
  - There is an allocated string for the text.
  - The string is not empty.
- Given some data in the field, `EditSaveData` 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.
- `EditSaveData` calls `MemHandleLock` to lock 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.

- `EditSaveData` calls `DmWrite`, which 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. This allows the memory system to 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 busy, meaning in use by the application. Releasing the record lets the database manager regain control of the record. In the call to `DmReleaseRecord`, the application also indicates to the database manager that the record contains "dirty" data; that is, data that has been changed.

Adding an Edit Button to the Main Form

To have an Edit button bring up the Edit form, you have to first create the resource and name it correctly so the code can respond to it.

If you remember button creation from previous phases, create one for the Main form, give it the following properties, and save the resource file:

Object Identifier	Edit
Button ID	(Assigned by Constructor)
Left and Top Origin	10, 146
Width and Height	40, 12
All buttons	Leave checked
Font	Standard
Label	Edit

Otherwise, follow these steps:

1. **In Constructor, open MemoPad.rsrc and double-click on Forms > Memo Pad Main in the Constructor project window.**

   Constructor opens the Form Editor for the Main form.

2. **In the Catalog window, select the Button icon and place it on the Form in the Layout Appearance panel.**
3. **Give the button the properties in the table above.**
4. **Save the resource file.**

Code Changes: Incorporation of Edit Button

This section briefly discusses the New Handler for the Edit Button and the Revised Handler for the New Button.

New Handler for the Edit Button

The `MainFormHandleEvent` routine now handles the new Edit button.

In the `case` for `MemoPadMainEditButton`:

- The `CurrentRecord` global variable is set to the index of the record to be edited. In this example, the record to be edited is always the first record in the database, index value zero.
- A call to `FrmGotoForm` initiates the transition to the Edit form. This makes the Edit form the active form.

Revised 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.

Code Changes: Text Retrieval From Database Record

In MemoPad 5, the Edit Form Handler has been revised to handle text retrieval.

- `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 `frmOpenEvent` 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:
  - Checks if there is a current record by checking the value of the `CurrentRecord` global variable.
  - Checks if the database contains any records.
  - Checks if the value of `CurrentRecord` is `noRecordSelected`. This happens when the New button is tapped on the Main form. In this case, no attempt is made to read from the database, and the field retains its initialized value, an empty string.

  If all checks are passed, that is, if a record is present, `EditRetrieveData` performs these actions:
  - A call to `DmGetRecord` gets a handle for the first record in the database. (`CurrentRecord` was set to zero in the code handling `MemoPadMainEditButton`.)
  - A call to `MemHandleLock` locks the handle to pin down the record in memory, and returns a pointer to the record. Locking down the handle is necessary before the data in the record can be directly manipulated.
  - A call to `MemHandleNew` allocates a new block of memory to contain a copy of the data for the field in the form. `MemHandleNew` is called with a size determined by the size of the data in the record. This size is the `StrLen` of the data, plus 1 for the terminator, which `StrLen` doesn't count. The function returns a handle to the new memory block.
  - Another call to `MemHandleLock` locks down the new memory block.

- A call to `StrCopy` copies the data from the first database record to the new memory block. 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.
- After the record's data has been retrieved, the record 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`.

Building and Downloading the Application

To build and download an application, follow these steps:

1. **Delete any previous MemoPad applications on the device and make sure it's correctly connected with your desktop computer.**

   See Deleting the MemoPad Application for more information.

2. **Open your project (.mcp file) in CodeWarrior.**
3. **Choose Project > Remove Object Code and OK the dialog.**
4. **Choose Project > Reset Project Entry Paths.**
5. **In the Project menu, check that Disable Debugger is displayed. If Enable Debugger is displayed instead, select it to enable the debugger.**

   The debugger should be enabled if you want to debug the application you're building for the device.

6. **Press F7 to build (Make) the executable.**

   CodeWarrior creates a new .prc file and also a new .prc.psym file.

7. **Double-click the newly created .psym file.**

   CodeWarrior prompts you to enter the "shortcut .2" sequence on the device.

8. **Turn on the device and enter the sequence: ♌ .. 2, then OK the dialog on the PC screen.**

   The application is downloaded to the device. A dialog box displays the progress.

   Next, the Debugger launches the program and stops execution at the first line of the `PilotMain` function.

9. **You can now debug the application on the device just as you would debug a resident application.**

   See the CodeWarrior documentation for more information.

Preparing the Device for Testing

1. **To exercise the application in standalone mode, choose File > Exit to quit the Debugger.**

   This should reset the Device.

2. **Remove the device from the cradle.**

3. **If exiting the Debugger hasn't reset the device, reset it by hand.**

   See Resetting the Device for more information.

4. **Choose the Applications silk-screened button and select the MemoPad (not Memo Pad) application in the launcher.**

Exercising the Application

To see whether your changes have taken effect, follow these steps:

1. **Tap the Main form's New button.**

   The application displays the Edit form

2. **Enter some words into the text field, like "Toto, now I know we're not in Kansas any more!".**

   Note: To change the location of the insertion point, tap the displayed text.

3. **Select the Done button.**

   The application returns to the Main form.

4. **Select the Edit button.**

   The Edit form is redisplayed. Note that the previously entered text is displayed again.

5. **Select the Done button.**

   The application returns to the Main form.

6. **Follow steps 1 - 5, entering some other text like "That's a horse of a different color".**

   Notice that the system is able to save the newly entered text.

Interacting With the PalmPilot Console Window

This section shows you how to use Console commands to interact with the application running on the device. The Debugger is discussed in more detail in the CodeWarrior documentation.

Note: Type `Help` for a complete list of all Console commands. The Palm OS Cookbook provides a complete reference to all console commands.

To use Console commands, follow these steps:

1. **With the Debugger running and connected to an application on the device, choose Console > PalmPilot Console.**

   The debugger displays the Console window.

2. **Select the Console window and press Enter (the Enter key on the numeric keypad).**

   You can type Ctrl-Return instead of using the Enter key.

   The insertion point moves to a new line.

3. **Type the command "opened" in the Console window 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.

4. **To examine information about the records in the MemoPad database, type "listrecords", a space, and the value of accessP for MemoPad's database, then press Enter.**

   For example:

   ```
   listrecords 00DAB484
   ```

   (It's convenient and less error prone to use copy and paste on access pointers.)

   A list of the current records appears, including the index, some of the data, and other information. Our current application's record list 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. **Choose File > Exit to quit the Debugger.**

Editing a Data Record in Place

Overview

Phase 6 introduces a technique for working with data records known as edit-in-place. The phase discusses these topics:

- About Edit-in-Place
- Code Changes: New Handler for Edit-in-Place

Phase 6 makes no changes to resources.

Setup

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

Note: If you've just completed Phase 5, you don't need to copy `MemoPad.mcp` or the two `MemoPad.rsrc` files.

Copy these files...	From...	To...
`MemoPad.c`	`Tutorial\MemoPad 06\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 05\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 05\Src\Resource.frk`	`Apps\Working Folder\Src\Resource.frk`
`MemoPad.mcp`	`Tutorial\MemoPad 05`	`Apps\Working Folder`

About Edit-in-Place

In the previous example (Phase 5), the program copies the data 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 to long term storage.

On traditional desktop systems, an application has to create an extra copy and move data back and forth because of the inefficiencies of editing directly in long-term storage.

The Palm OS system, however, 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. Moving data from one area of dynamic memory to another for presentation and editing in the user interface is not required.

This means the user, through the user interface, can edit the data in a database record directly, while it is still in the database. The program doesn't have 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.

Code Changes: New Handler for Edit-in-Place

The file `Phase06:Src:MemoPad.c` includes code changes required to set up an edit-in-place mechanism. They include:

New CreateRecord Routine

`MainFormHandleEvent` contains a `MemoPadMainNewButton` case. It creates a new record in the database by calling the 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` performs the following actions:

- A call to `DmNewRecord` creates a new record in the application's database. The new record has a default size of `newMemoSize`. The data in the record is initialized to an empty string.
- A call to `MemHandleLock` locks the new record data and gets a pointer.
- The call to `DmWrite` writes a single zero byte to the first position in the record. `DmWrite` returns a value that `error` is set to. Then, `ErrFatalDisplayIf` checks the error and, if it's non-zero, displays an alert message box containing the specified error message. This alert doesn't let the program continue and should only be used if 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, a call to `MemPtrUnlock` unlocks it.

- A call to `DmReleaseRecord` 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; that is, data has been changed.
- Finally, the global variable `CurrentRecord` is set to the index of the first record, where this new record was created.

Modified Routines

Phase 6 contains modified routines for `EditRetrieveData` and `EditSaveData`:

- `EditRetrieveData` has been simplified. Once the current record has been retrieved by a call to `DmGetRecord`, the record handle 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 the Phase 05 MemoPad that has been replaced is included for reference.)
- `EditSaveData` has been simplified. 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's important to clear the handle reference in the field data structure: When the Edit form is closed, the form is disposed of and cleared from memory. Part of that process frees all memory blocks associated with 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.

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

Adding a List for Record Selection and Display

Overview

This phase adds a scrolling list to the Main form. The 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 user taps an item, the Edit form displays the entire text of the corresponding record in the text field.

The phase discusses the following topics:

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

Setup

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

Note: If you've just completed Phase 6, you don't need to copy `MemoPad.mcp` or the two `MemoPad.rsrc` files.

Copy these files...	From...	To...
`MemoPad.c`	`Tutorial\MemoPad 07\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 06\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 06\Src\Resource.frk`	`Apps\Working Folder\Src\Resource.frk`
`MemoPad.mcp`	`Tutorial\MemoPad 06`	`Apps\Working Folder`

Adding a Record Display List to the Main Form

In this section, you'll make the following changes to the Main form:

- Adding a List to the Main Form
- Removing the Edit and Exit Buttons

Adding a List to the Main Form

To add a list to the Main form, follow these steps:

1. **In Constructor, open MemoPad.rsrc and double-click on Forms >Memo Pad Main in the Constructor project window.**

   Constructor opens the Form Editor for the Main form.

2. **In the Catalog window, select the List icon and place it on the Form in the Layout Appearance panel.**

3. **Give the list the following properties:**

Object Identifier	Memo list
List ID	(assigned by Constructor)

Left and Top Origin	10, 17
Width	140
Font	Standard
Visible Items	11
List Items	(leave empty)

Removing the Edit and Exit Buttons

With Constructor, removing buttons is even easier than adding them. To remove the Edit and Exit buttons, follow these steps:

1. **Display the Main form, select the Edit button, and press the Backspace key on the keyboard.**
2. **Repeat for the Exit button.**

   Both buttons are removed from the UI that this form presents.
3. **Save the resource file.**

Code Changes: Using a List of Records

Adding or removing a resource required changes to code. In particular, the code for this phase had to be revised for the removal of the Exit button and for the inclusion of list handling. This section walks you through these changes. It discusses:

- Revised #define Macros
- The MainFormInit Function
- Other Changes

Revised #define Macros

The `MemoPadRsc.h` file has three changes to the previous version:

- A new `#define` macro to the `MemoPadRsc.h` file: `mainList` identifies the list resource to the application code.
- Because the Exit button resource is gone, so is its macro.
- Because the Edit button resource is gone, so is its macro.

Constructor takes care of these changes automatically.

The MainFormInit Function

For this version, `MemoPad.c` has been revised to have a list of database records appear when the Main form opens.

To do that, `MainFormHandleEvent` now calls a new routine `MainFormInit`, to set up the list and draw the Main form, when the `frmOpenEvent` case activates.

`MainFormInit` initializes the Main form and the list. It performs the following actions:

`MainFormInit` first calls `DmNumRecords` to check whether there are any records in the database. If there are none, no list setup is needed, but the form is drawn.

If the database contains records, `MainFormInit` must set up the list, so that each database record is represented by one item. To do that, `MainFormInit`:

- Gets the usable width of the list rectangle.

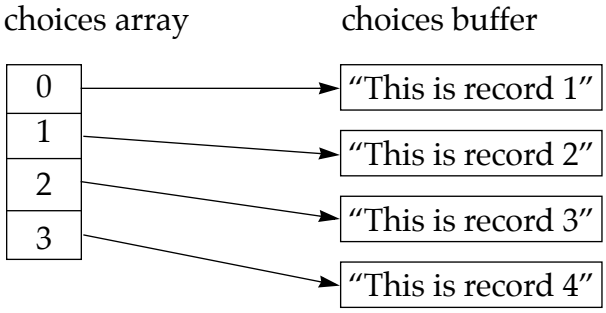
  The width of the list's display rectangle limits the number of characters displayed for an item.
- Allocates an initial block for the list choices, locks it down, and sets up its initial value.
- Builds up the choices by looping though the items (records) and doing the following:

MainFormInit calls:	To achieve this:
`DmGetRecord`	Retrieve the record.
`MemHandleLock`	Lock the record in place.
`StrLen`	Determine the length of the record text that fits within the list bounds.

MainFormInit calls:	To achieve this:
`FntCharsInWidth`	Determine how many characters of the string can fit within the bounds of the list rectangle. `MainFormInit` gets a pointer to the list, gets the width (in pixels) of the list's rectangle from the list data structure, and decreases this width by a couple of pixels to ensure that the text doesn't overlap with the frame of the list.
`MemHandleResize`	Increase the size of the `choices` buffer to contain a copy of the text from the record. (The `choices` buffer is an array of packed strings.) Copy the data from the database record to the current location (`choicesOffset`) in the choices buffer. Place a zero terminator at the end of the copied text. Update the `choicesOffset` variable to the location for the next record text.
`MemHandleUnlock`	Unlock the handle.
`DmReleaseRecord`	Release the record.

- The result of the `MainFormInit` loop is a memory block with packed strings in it. `MainFormInit` now calls `SysFormPointerArrayToStrings` to construct 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. The handle must be global, because it will be freed just before the Main form is closed:



- `MainFormInit` calls `LstSetListChoices` to pass the items prepared in the previous step to the system's list processing code to be displayed in the list.
- `MainFormInit` calls `FrmDrawForm` to draw the form with the list in it.

Other Changes

Revised MainFormHandleEvent Routine

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.

The `handled` variable is not set to `TRUE` for this event. The system's form-handling code needs to process this event as part of its own form cleanup.

Deleted Handler for the Exit and Edit Buttons

The Exit button is not needed because the user doesn't explicitly exit in a typical Palm OS application. Instead, the user simply launches another application. When this happens, the system automatically

adds an `appStopEvent` to the currently executing application's event queue.

New 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.

Code Changes: Adding Multiple Records to the Database

The previous code changes almost make it possible to add multiple records to the database (though some additional changes had to be made for this phase).

When the user taps the New button on the Main form, the program calls the `CreateRecord` routine. This routine always inserts a new record as the first record before the previous first record in the database.

The list set up in `MainFormInit` can 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. Empty 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 the user creates a new record and doesn't enter any data.
- If the user completely deletes the text of an existing record.

After `EditSaveData` retrieves the pointer to the text data, it determines whether text is present. If the current database record is empty, a call to `DmRemoveRecord` removes it from the database.

Building and Downloading the Application

To build and download an application for use with the CodeWarrior Debugger, follow these steps:

1. **Delete any previous MemoPad applications on the device and make sure it's correctly connected with your desktop computer.**

   See Deleting the MemoPad Application for more information.

2. **Open your project (.mcp file) in CodeWarrior.**
3. **Choose Project > Remove Object Code and OK the dialog.**
4. **Choose Project > Reset Project Entry Paths.**
5. **In the Project menu, check that Disable Debugger is displayed. If Enable Debugger is displayed instead, select it to enable the debugger.**

   The debugger should be enabled if you want to debug the application you're building for the device.

6. **Press F7 to build (Make) the executable.**

   CodeWarrior creates a new .prc file and also a new .prc.psym file.

7. **Double-click the newly created .psym file.**

   CodeWarrior prompts you to enter the "shortcut .2" sequence on the device.

8. **Turn on the device and enter the sequence: ♌ .. 2, then OK the dialog on the PC screen.**

   The application is downloaded to the device. A dialog box displays the progress.

   Next, the Debugger launches the program and stops execution at the first line of the `PilotMain` function.

9. **You can now debug the application on the device just as you would debug a resident application.**

   See the CodeWarrior documentation for more information.

Preparing the Device for Testing

1. **To exercise the application in standalone mode, choose File > Exit to quit the Debugger.**

   This should reset the Device.

2. **Remove the device from the cradle.**

3. **If exiting the Debugger hasn't reset the device, reset it by hand.**

   See Resetting the Device for more information.

4. **Choose the Applications silk-screened button and select the MemoPad (not Memo Pad) application.**

Exercising the Application

With the application running; follow these steps:

1. **Select the Main form's New button.**

   The Memo Pad application displays the Edit form.

2. **Enter several words into the text field, for example "Follow, follow, follow, follow, follow the yellow brick road".**

   If you'd like to use the keyboard shortcut to copy and paste some of the "follow"s, draw the shortcut character into the Graffiti area with the mouse.

3. **Select the Done button.**

   The Memo Pad application returns to the Main form. The Main form now displays a list with a single item at the top already selected.

4. **Select the item in the list.**

   The Memo Pad application displays the Edit form again. Note that the previously entered text is still there.

5. **Select the Done button to return to the Main form.**

6. **Select the Main form's New button.**

7. **Write with Graffiti several words, such as "There's no place like home" to appear in the text field and select 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's truncated to fit within the horizontal bounds of the list rectangle.

8. **Select one of the two items and delete all its text in the Edit form that appears.**

9. **Select the Done button to return to the Main form.**

   The Palm OS system has removed the item from the list and from the database.

10.

Displaying List Items As Required

Overview

Phase 8 describes an alternative to packed strings for delivering display items, "display-as-required".

The phase discusses these topics:

Setup

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

Note: If you've just completed Phase 7, you don't need to copy `MemoPad.mcp` or the two `MemoPad.rsrc` files.

Copy these files...	From...	To...
`MemoPad.c`	`Tutorial\MemoPad 08\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 07\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 07\Src\Resource.frk`	`Apps\Working Folder\Src\Resource.frk`
`MemoPad.mcp`	`Tutorial\MemoPad 07`	`Apps\Working Folder`

The Display-As-Required Approach

Phase 7 allows the user to display 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. Each record in the database is represented by an item in the list.

There are problems with this approach. If a database contains a large number of records, the corresponding collection of packed strings could exhaust available dynamic memory. In addition, the approach requires 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 applications don't have to provide an array of packed strings to the list processing code. Instead, an 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's required.

Code Changes: Drawing List Items

Phase 8 of `MemoPad.c` incorporates changes for drawing the list items as required (No changes have been made to the resources from the Phase 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 is responsible for drawing the items in the list.

`LstDrawList` calls `MainFormListDrawItem` once for each item that is displayed in the list. Note that this includes only those items that are currently displayed in the list, not every item that might be shown. When the list is scrolled, `LstDrawList` calls `MainFormListDrawItem` once for each item that scrolls into view.

`MainFormListDrawItem` is passed:

- The item number, which in this example is interpreted as the record number.
- The display bounds within which the list item should be drawn.
- A pointer to data (not used in this example).

`MainFormListDrawItem`'s task is to draw the data for this item within the given bounds. `MainFormListDrawItem`:

- Retrieves the record with index `itemNum` retrieved from the database and locks it down.
- Calculates the length of the text to be drawn, using the `bounds` argument to constrain the width.
- Calls `WinDrawChars` to draw the characters on the display, starting at the top-left corner of the bounds.
- Unlocks the record and releases it back to the system.

Adding a Page Menu to the Edit Form

Overview

This phase adds a Page menu with New Page and Delete Page commands to the Edit form.

- 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.

You first create the menu resources and then examine the code that adds and deletes MemoPad pages.

The phase discusses these topics:

Note: If you've just completed Phase 8, you don't need to copy `MemoPad.mcp` or the two `MemoPad.rsrc` files.

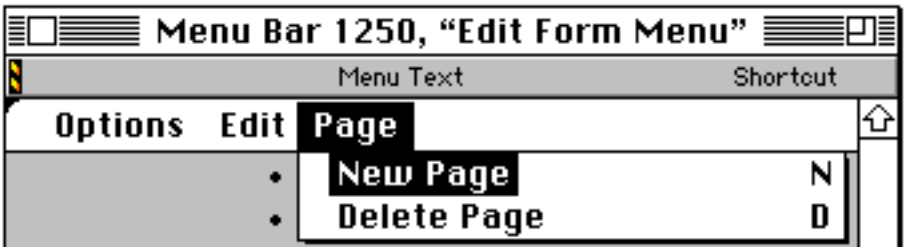
Copy these files...	From...	To...
`MemoPad.c`	`Tutorial\MemoPad 09\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 08\Src`	`Apps\Working Folder\Src`

Copy these files...	From...	To...
MemoPad.rsrc	Tutorial\MemoPad 08\ Src\Resource.frk	Apps\Working Folder\ Src\Resource.frk
MemoPad.mcp	Tutorial\MemoPad 08	Apps\Working Folder

Creating the Page Menu

This section shows you how to add a Page menu to the Edit form. The menu has two items, New Page and Delete Page.

The process of adding a Page menu resembles that for the Options menu in Phase 2. The task is to add a menu that looks like the following:



If you're already familiar with the process, feel free to add the menu without instruction.

Otherwise, follow these steps:

1. **In Constructor, open MemoPad.rsrc.**
2. **In the Constructor project window, open Menu Bars and double-click on Edit Form Menu.**

   Constructor displays the Menu Editor.
3. **In the Constructor Project window, select "Menus" and type Ctrl-K.**

4. **Name the newly created menu "Edit Page" and drag the icon into the Menu Editor to be the last menu of the Edit form in the menu bar.**
5. **Change the menu text to "Page".**
6. **With Page still selected, type Ctrl-K to add the menu item "New Page", then tab to the shortcut and enter "N".**

   You have to make sure the window is wide enough to see the shortcut.
7. **Type Ctrl-K again to add the menu item "Delete Page", then tab to the shortcut and enter D.**

   You should now have the menu in the picture above step 1.
8. **Close the Menu Editor and save the resource file.**

   When you save the .rsrc file, Constructor automatically generates the appropriate header file.

Adding a Resource for a Delete Memo Alert

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

1. **In the Constructor project window, highlight Alerts and type Ctrl-K.**

   Constructor creates an alert instance.

2. **Name the alert instance "Delete Alert", then double-click it.**

   Constructor displays the Alert Editor.

3. **Set the values for this alert as follows:**

Alert Type	Confirmation
Default Button	(Change after Step 5)
Title	Delete Memo
Message	Do you wish to permanently remove this page?
Item Text 0	OK

4. **To get a second button, select Item Text 0 in the Alert Properties panel of the Alert Editor, then choose Ctrl-D (Edit > Duplicate Button Title).**

5. **Still in the Alert Properties panel, change the text of the new button to Cancel.**

6. **Set Default Button to 1.**

   This means the Cancel button will be the default. The button number is included in the button title after "Item Text", for example, "Item Text 1".

7. **Close the Alert Editor and save the resource file.**

Code Changes: Handlers for the New Commands

`MemoPad.c` includes changes required for the new menu commands. There are two new cases, `EditPageNewPage` and `EditPageDeletePage`, in the `EditDoMenuCommand` routine.

`EditPageNewPage` follows these steps to create a new menu page:

- `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.

`EditPageDeletePage` follows these steps to delete a memo page:

- Displays an Alert dialog to ensure it's the user's intention to delete a page:
  - `FrmAlert` retrieves the ID of the delete alert, 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 has been tapped, any text in the field is deleted. The function deletes the text in the field, then returns to the Main form. This then triggers the usual `EditSaveData` sequence and, since the field is empty, the record is deleted.

Building and Downloading the Application

To build and download an application for use with the CodeWarrior Debugger, follow these steps:

1. **Delete any previous MemoPad applications on the device and make sure it's correctly connected with your desktop computer.**

   See Deleting the MemoPad Application for more information.

2. **Open your project (.mcp file) in CodeWarrior.**
3. **Choose Project > Remove Object Code and OK the dialog.**
4. **Choose Project > Reset Project Entry Paths.**
5. **In the Project menu, check that Disable Debugger is displayed. If Enable Debugger is displayed instead, select it to enable the debugger.**

   The debugger should be enabled if you want to debug the application you're building for the device.
6. **Type F7 to build (Make) the executable.**

   CodeWarrior creates a new .prc file and also a new .prc.psym file.
7. **Double-click the newly created .psym file.**

   CodeWarrior prompts you to enter the "shortcut .2" sequence on the device.
8. **Turn on the device and enter the sequence: ♌ .. 2, then OK the dialog on the PC screen.**

   The application is downloaded to the device. A dialog box displays the progress.

   Next, the Debugger launches the program and stops execution at the first line of the `PilotMain` function.
9. **You can now debug the application on the device just as you would debug a resident application.**

   See the CodeWarrior documentation for more information.

Preparing the Device for Testing

1. **To exercise the application in standalone mode, choose File > Exit to quit the Debugger.**

   This should reset the device.
2. **Remove the device from the cradle.**
3. **If exiting the Debugger hasn't reset the device, reset it by hand.**

   See Resetting the Device for more information.
4. **Choose the Applications silk-screened button and select the MemoPad (not Memo Pad) application.**

Exercising the Application

With the application running, follow these steps to exercise the application:

1. **Select the New button.**
2. **Type some text into the new page.**
3. **Choose the Page > New Page menu command.**

   The MemoPad application adds another page for editing.
4. **Type some text into the new page and select the Done button.**
5. **From the Main form List View, select one of the records.**
6. **Choose the Page > Delete Page menu item and respond to the Delete alert dialog.**

   Depending on your response, the MemoPad application deletes the record or keeps it.

Adding a Details Dialog and the Secret Record Attribute

Overview

In Phase 10, you add a Details button, a Details dialog, and a secret record attribute.

Palm OS allows users to mark records as secret (private) and then turn off display of secret records until a password is entered. In this phase, you add a button to be used as the interface, then look at the code that implements that behavior. The phase discusses the following topics:

Setup

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

Note: If you've just completed Phase 9, you don't need to copy `MemoPad.mcp` or the two `MemoPad.rsrc` files.

Copy these files...	From...	To...
`MemoPad.c`	`Tutorial\MemoPad 10\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 09\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 09\Src\Resource.frk`	`Apps\Working Folder\Src\Resource.frk`
`MemoPad.mcp`	`Tutorial\MemoPad 09`	`Apps\Working Folder`

Adding a Details Dialog to the Edit Form

In this section, you add a Details button to the Edit form and a Details dialog (a new form) to the project. When the phase is complete, the user can tap the Details button to display the Details dialog.

Adding Details Button to the Edit Form

This section adds a Details button to the Edit form. The button will have the following values:

Object Identifier	Details
Button ID	(Assigned by Constructor)
Left and Top Origin	43, 147
Width and Height	36, 12
All checkboxes	Leave checked

Font	Standard
Label	Details...

If you already know how to add a button to the Edit form, do so and use the values in the table above. Otherwise, follow these steps:

1. **With Constructor running an MemoPad.rsrc open for editing, click on Forms in the Constructor project window, then double-click on Memo Pad Edit.**

   Constructor displays the Edit form in the Form Editor.

2. **In the Catalog window, select a Button icon and drag it onto the Edit form.**

3. **Set the values in the Layout Properties window to the ones shown in the table above.**

Creating a Details Form

The Details form will be a new modal dialog that allows users to delete records or mark them as secret. Figure 10.1 shows the details dialog after all changes have been made.

To create the Details form with three buttons, follow these steps:

1. **With Constructor running, highlight Forms in the Constructor project window and type Ctrl-K.**

   Constructor adds a new form.

2. **Change the name of the form to Memo Pad Details, then double-click on the icon.**

   Constructor displays the Details form.

3. **Set the Form's Layout Properties as follows:**

Left and Top Origin	2, 86
Width and Height	156, 72
Usable, Modal, Save Behind	Checked

Default Button ID	0
Form Title	Details

4. **Drag three buttons from the Catalog window onto the Details form displayed in the Layout Appearance panel. Give the buttons the following properties:**

OK button:

Object Identifier	OK
Button ID	(signed by Constructor)
Left and Top Origin	5, 55
Width and Height	36, 12
All checkboxes	Leave checked
Font	Standard
Label	OK

Cancel button:

Object Identifier	Cancel
Button ID	(Assigned by Constructor)
Left and Top Origin	48, 55
Width and Height	36, 12
All checkboxes	Leave checked
Font	Standard
Label	Cancel

Delete button:

Object Identifier	Delete
Button ID	(Assigned by Constructor)
Left and Top Origin	91, 55
Width and Height	36, 12
All checkboxes	Leave checked
Font	Standard
Label	Delete...

5. **Drag a Label icon from the Catalog to the Layout Appearance panel.**

   This label will be used for the “secret” checkbox.

6. **Give the Label the following values:**

Object Identifier	Private
Label ID	(assigned by Constructor)
Left and Top Origin	13, 33
Usable	Leave checked
Font	Choose **Bold** from the popup menu.
Text	Private:

7. **Drag a checkbox icon from the Catalog to the Details form.**

   This checkbox will be used to mark a record as secret.

8. **Give the checkbox the following properties:**

Object Identifier	Private
Checkbox ID	(Assigned by Constructor)
Left and Top Origin	58, 32
Width and Height	14, 14
Usable	Checked
Selected	Unchecked
Group	0
Font	Standard
Label	(Remove any text. Auto-labels are displayed to the right of the checkbox.)

9. **Save the resource file**

A Note on Naming Conventions

You may have noticed that some of the names for the buttons in this Details form were the same as the button names on other forms. This won't create problems for the code, however, because Constructor combines the form name and button name when generating the header file, then adds the object type to create uniquely-named resources. For example, MemoPadDetailsOKButton.

In contrast to Catalog resources, all resources in the Constructor project window have to have unique names.

Figure 10.1 Layout Appearance of the Details form

Adding a Help String to the Details Form

A Help string provides additional information about a modal form to the user. Note that Constructor is designed to only allow Help strings for modal forms.

To add a Help string for the Details form, follow these steps:

1. **If Memo Pad Details form isn't displayed in a Form Editor, double-click on it the Constructor project window.**
2. **Assign the number 1305 as the Help ID, then click on Create.**

   Constructor displays the String Editor.
3. **Type in:**

   ```
   To hide private records, tap the Application's
   (arrow) icon and go to the Security application.
   ```

   then close the String Editor.
4. **In the Constructor project window, name the newly created string Details:Help.**
5. **Save the resource file.**

Code Changes: New Event Handlers

This phase adds event handlers for the Details button and for the Details dialog.

Details Button Event Handler

The `EditFormHandleEvent` routine has been modified to handle the new Details button. There is now a test for `editDetailsButton`.

When the button is tapped, a call to `FrmPopupForm` starts the Details form. `FrmPopupForm` generates a `frmLoadEvent` and a `frmOpenEvent`. It does not generate a `frmCloseEvent` before the other two events. As a result, the Edit form is not erased before the Details form is displayed. This has the effect of popping up the Details form on top of the Edit form.

Details Dialog Event Handler

The `ApplicationHandleEvent` routine has been modified to include a case for the Details form. It calls `FrmSetEventHandler` so that the application's new `DetailsFormHandleEvent` routine receives events while the Details form is active.

`DetailsFormHandleEvent` processes events for the Details form. It includes:

- Processing for each of the three buttons on the form.
- Processing for the `frmOpenEvent.`

For the `frmOpenEvent`, `DetailsFormHandleEvent:`

- Calls the new `DetailsInit` routine to set up the form.
- Calls `FrmDrawForm` to display the form is displayed.

The `DetailsInit` routine:

- Retrieves the setting for "private" for the current record.
- Sets the corresponding value for the private checkbox in the form.
- Calls `DmRecordInfo` to retrieve 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 `DetailsFormHandleEvent` routine contains a switch to handle the `ctlSelectEvent` for each button on the form.

- **OK Button—**The case for the OK button:

  Calls the new `DetailsApply` routine to apply changes made in the Details form to the current record.

  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.

  Clears the Details form and calls `FrmReturnToForm` to display the Edit form. `FrmReturnToForm` erases and deletes the Details form and then activates the Edit form. `FrmReturnToForm` assumes that the Edit form is already loaded into memory.

  If the checkbox value is different from the setting in the record attribute, the user changed the checkbox on the Details form. In that case:

  – The new value for the `secret` flag is set in the `attribute` byte. (A call to the new routine `GetObjectPtr` gets a pointer to the private checkbox's data.)

  – The `dirty` flag is set in the `attribute` byte. This flag indicates to the database manager that the record contains data that has changed.

  – The updated `attribute` byte is written to the database by a call to `DmSetRecordInfo`.

- **Cancel button—**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.

- **Delete button—**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 that the case for the deletePage command in `EditDoMenuCommand` has been revised to call the `EditDeleteCurrentRecord` routine.

Building and Downloading the Application

To build and download an application for use with the CodeWarrior Debugger, follow these steps:

1. **Delete any previous MemoPad applications on the device and make sure it's correctly connected with your desktop computer.**

   See Deleting the MemoPad Application for more information.

2. **Open your project (.mcp file) in CodeWarrior.**
3. **Choose Project > Remove Object Code and OK the dialog.**
4. **Choose Project > Reset Project Entry Paths.**
5. **In the Project menu, check that Disable Debugger is displayed. If Enable Debugger is displayed instead, select it to enable the debugger.**

   The debugger should be enabled if you want to debug the application you're building for the device.

6. **Press F7 to build (Make) the executable.**

   CodeWarrior creates a new .prc file and also a new .prc.psym file.

7. **Double-click the newly created .psym file.**

   CodeWarrior prompts you to enter the "shortcut .2" sequence on the device.

8. **Turn on the device and enter the sequence: ♌ .. 2, then OK the dialog on the PC screen.**

   The application is downloaded to the device. A dialog box displays the progress.

   Next, the Debugger launches the program and stops execution at the first line of the `PilotMain` function.

9. **You can now debug the application on the device just as you would debug a resident application.**

   See the CodeWarrior documentation for more information.

Preparing the Device for Testing

1. **To exercise the application in standalone mode, choose File > Exit to quit the Debugger.**

   This should reset the device.

2. **Remove the device from the cradle.**

3. **If exiting the Debugger hasn't reset the device, reset it by hand.**

   See Resetting the Device for more information.

4. **Choose the Applications silk-screened button and select the MemoPad (not Memo Pad) application in the launcher.**

Exercising the Application

With the application up and running, follow these steps to test the new features:

1. **Select the New button on the Main form.**

   The application displays the Edit form.

2. **Enter a few words into the text field.**

3. **Select the Details button.**

   The application displays the Details dialog.

4. **Select the Delete button.**

   The Confirmation Alert appears.

5. **Choose Cancel to return to the Details dialog.**

6. **Select the Delete button again.**

   The Confirmation Alert appears again.

7. **Choose OK.**

   The Details and Edit forms are erased and the Main form is displayed with no records in the list.

Adding Categories

Overview

In Phase 11, you add category assignment to the memos. The Main form is enhanced to show all the records (as before) or show only those records in a single category. This overview first provides some background information, then discusses Categories in MemoPad and finally lists the topics discussed in this phase.

What Are Categories?

Categories are a fundamental element of Palm OS databases. Under Palm OS, categories serve a similar function to folders on a desktop computer. Record-based Palm OS applications generally offer category-based viewing of records in the list view.

A standard record type 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.

Applications usually allow the user to create and edit the categories, but they may also preload a fixed set of categories. Applications also generally allow users to assign records to categories as they wish; this Tutorial chooses this approach. Alternatively, applications can assign records internally based on program logic.

Categories in MemoPad

In this phase, the Edit and Details forms are enhanced to display and assign the category of the current record. A new dialog is added to allow the user to create and edit category names.

The phase discusses the following topics:

Setup

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

Note: If you've just completed Phase 10, you don't need to copy `MemoPad.mcp` or the two `MemoPad.rsrc` files.

Copy these files...	From...	To...
`MemoPad.c`	`Tutorial\MemoPad 11\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 10\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 10\Src\Resource.frk`	`Apps\Working Folder\Src\Resource.frk`
`MemoPad.mcp`	`Tutorial\MemoPad 10`	`Apps\Working Folder`

Adding the Category UI to the Main Form

This section steps you through creating a category trigger and a popup selection list on the Main form.

1. **With Constructor running, open MemoPad.rsrc.**
2. **Display the Memo Pad Main form in the Form Editor.**
3. **In the Catalog window, select a List icon and drag it onto the form.**

   This list is displayed by the program when the user selects a popup trigger.
4. **Set the values for the List as follows:**

Object Identifier	Categories List
List ID	(Filled in by Constructor)
Left Origin, Top Origin	86, 1
Width	72
Usable	**Unchecked** (This list should not be displayed by default.)
Font	Standard
Visible Items	0

5. **Note the List ID, which you will need later to join the list and the Popup Trigger.**
6. **In the Catalog window, select a Popup Trigger icon and drag it on top of the list.**
7. **Set the values in the Layout Properties window as follows.**

Note: You must set the width before the left origin, making the popup list invisible in Constructor. The program will take care of making the list fit precisely.

Object Identifier	Category
Popup ID	(Filled in by Constructor)
Left Origin and Top Origin	160, 0 (do this last!)
Width and Height	0, 13
Usable	Checked
Anchor Left	Unchecked
Font	Standard
Label	(Remove any text)
List ID	Enter the ID of the list you just created here (see "A Note on Popup Triggers" below)

8. **Close the Memo Pad Main Form Editor and save the file.**

A Note On Popup Triggers

Because you set the width of the popup trigger to 0, it isn't visible in the Constructor display. If, however, you need to make changes to the popup trigger object, you can select the form and choose Layout > Show Object Hierarchy.

Even though the trigger isn't currently visible, it becomes visible when you actually run the application. The code determines how far to the left of the right margin the trigger has to be, based on its label's width.

Adding the Category UI to the Edit Form and to the Details Form

Adding the Category UI to the MemoPad Edit form and MemoPad Details form is very similar to adding it to the Main form.

Edit Form

To add the Category UI to the MemoPad Edit form, follow these steps:

1. **Select and display the MemoPad Edit form.**
2. **Follow the same steps you did for the MemoPad Main form.**

   You can use the same name for the list and the popup trigger. When Constructor updates the header file, it creates a unique name by combining the name of the form and the name of the resource.
3. **Close the Memo Pad Edit Form Editor and save the resource file.**

Details Form

To add the Category UI to the Details form, you'll add a popup trigger and list and a label. Note that the trigger doesn't display under Constructor because the width is 0 and will be calculated by the program.

1. **Follow the steps you did for the Main form but use these values:**

   List:

Object Identifier	Categories List
List ID	(Filled in by Constructor)
Left Origin, Top Origin	59, 19

Width	72
Usable	**Unchecked** (This list should not be displayed by default.)
Font	Standard
Visible Items	0

Popup Trigger:

Object Identifier	Category
Popup ID	(Filled in by Constructor)
Left Origin and Top Origin	57, 18(do this last!)
Width and Height	0, 13
Usable	Checked
Anchor Left	**Checked**
Font	Standard
Label	(Remove any text)
List ID	Enter the ID of the list you just created here.

2. **Drag a Label icon onto the Details form and give it the following values:**

Object Identifier	Category
Label ID	(Filled in by Constructor)
Left Origin and Top Origin	4, 18
Usable	Checked
Font	**Bold**
Text	Category:

3. **Close the Memo Pad Details Form Editor and save the resource file.**

Adding the String List for the Categories

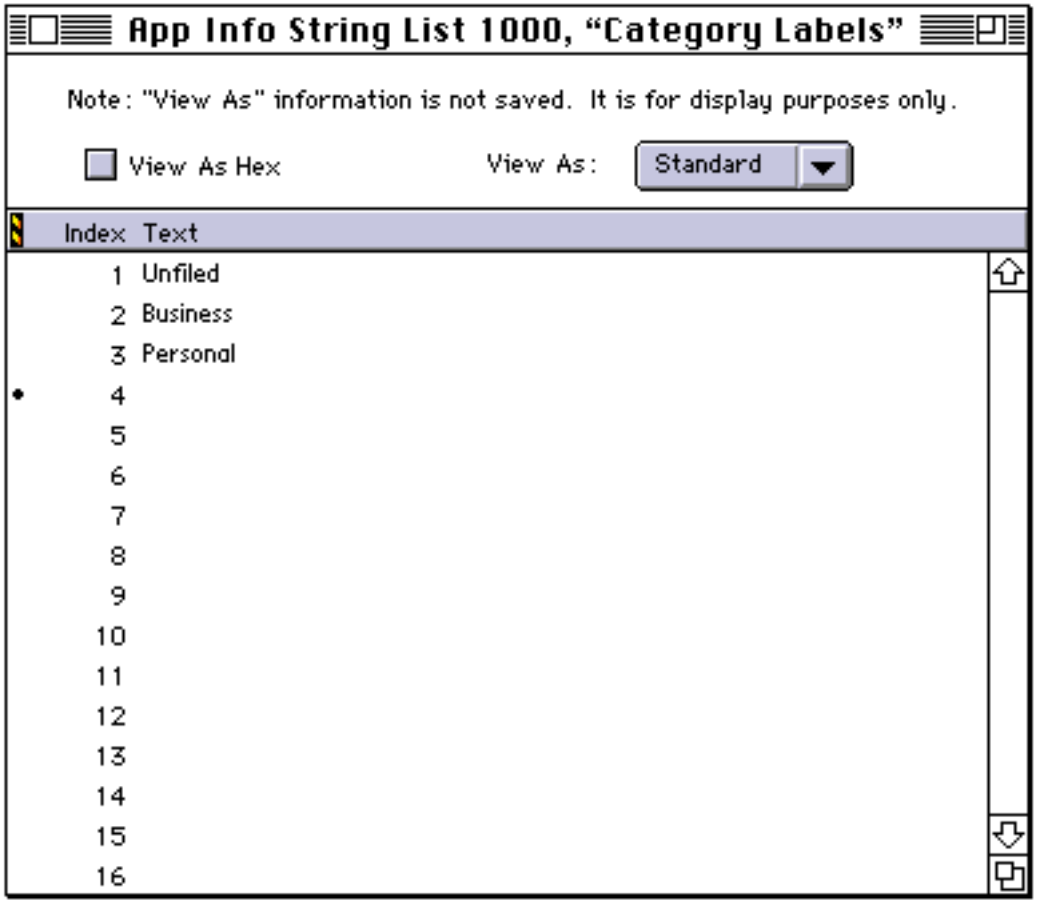
The only piece that's missing now is the default list of strings that will be popped up by the system when the user selects the Category popup trigger in any of the forms.

To create this string list, follow these steps:

1. **In the Constructor project window, select App Info String Lists.**
2. **Type Ctrl-K to create an instance of App Info String Lists.**
3. **Name the newly created list Category Labels and double-click on it.**

   The String List Editor appears.
4. **To have the list display properly, enter three text items, Unfiled, Business, and Personal, into the first 3 empty lines, then press Ctrl–K 12 times to create a total of 16 lines.**

   The String List Editor should look as follows:



5. **When you've entered all categories, close the String List Editor and save the resource file.**

Code Changes: Categories

This section provides background information on System Category Handling and The AppInfo Chunk. The rest of the chapter discusses category handling for the different forms:

- Category Handling for the Main View
- Category Handling for the Edit Form
- Category Handling for the Details Dialog

System Category Handling

The system provides a user interface component that allows the user to display, select, and edit category names. This component frees the application from most of the work associated with managing category names.

This phase adds event handling for the categories in the Main, Edit, and Details forms.

The AppInfo Chunk

`MemoPadInitAppInfo` creates an `AppInfo` chunk if there is none, and loads the default catagory strings from the applications App Info String List (tAIS) resource.

`The AppInfo` chunk is used to store category information. Category information is always placed in the beginning of the AppInfo block, where the `Category.c` system code expects it.

Category Handling for the Main View

To handle categories for the Main view, Phase 11 has two new functions and three changed functions:

- `MainViewLoadRecords` is a new function that loads the list object; it:
  - Gets the number of records currently in the application database in the current category.
  - Sets up the list with the items in the current category.

  This functionality has been extracted into one routine so it's accessible from several places.
- `MainViewSelectCategory` is a new function that handles selection, creation, and deletion of categories for the Main view.
- `MainListDrawItem` has been changed to find the category records to draw, skipping records not in the current category. This function is a callback that's invoked when the application calls `LstDrawList`.
- `MainViewInit` has been changed to set the label of the category trigger.
- `MainViewHandleEvent` has been changed to pop up the list of categories when the category trigger is pressed.

This section explains how the functions interact, discussing the following topics:

- Application Initialization and Shutdown
- Main View Initialization
- Responding to User Input

Application Initialization and Shutdown

To have categories readily available, there's some system setup when `StartApplication` is called:

- `StartApplication` calls `MemoPadInitAppInfo` when a new database is created.
- `MemoPadInitAppInfo`
  - Creates an `appInfo` chunk if it doesn't exist.

  - Installs default category names in the newly allocated `appInfo` block.

    This is a standard technique for storing category names. The the system provides functions and user interface based on this design.

- `StartApplication` 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.

To close any open forms, `StopApplication` calls `FrmCloseAllForms`.

Main View Initialization

Some additional initialization is performed by the `MainViewInit` routine to have the appropriate category trigger visible and all category names ready for display:

- `MainViewInit` calls the system routines `CategoryGetName` and `CategorySetTriggerLabel` to set the category list trigger name to the name of current category.
- `MainViewInit` calls `MainViewLoadRecords` to load the list with the records to display.
- In the `MainViewLoadRecords` routine, a call to `DmNumRecordsInCategory` replaces the call to `DMNumRecords`. Only records in the category the user selected are displayed.

Responding to User Input

When the user selects a category, the application has to respond appropriately as follows:

- `MainViewHandleEvent` calls `MainViewSelectCategory` to pop up the category list when the category trigger is pressed.
- `MainViewSelectCategory` calls the system function `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.
  - Draws the new list content by calling `LstDrawList` which in turn calls `MainViewListDrawItem`.
- In `MainViewListDrawItem`, the routine `DmSeekRecordInCategory` is now used because it will skip records in other categories.

  Using `DmSeekRecordInCategory` creates a potential performance problem if the number of records is large. To find the requested record, `DmSeekRecordInCategory` must start from record zero and advance forward n records while skipping records from other categories. This will be slow 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.

Category Handling for the Edit Form

Just like the Main form, the Edit form has two new and several revised functions:

- `EditViewUpdateDisplay` is a new function that updates the display of the Edit form.
- `EditViewSelectCategory` is a new function that changes the category of the memo in the Edit form.
- `EditViewRetrieveData` has been revised to:
  - Set the global variable that keeps track of the current category to the category of the current record.
  - Set the label that contains the note's category.
- `EditViewHandleEvent` has been revised to:
  - Pop up the category list when the category trigger is pressed.
  - Call `EditViewSelectCategory`.
  - Set `handled` to `TRUE`.
  - Have a new case `frmUpdateEvent`: If `handled` was set to `TRUE` by the code that pops up the category trigger, call `EditViewUpdateDisplay`. The `frmUpdateEvent` is added to the event queue from the Details dialog.

Category Handling for the Details Dialog

There are changes to functions that handle the Details dialog:

- `DetailsInit` now:
  - Initializes to the current record's current category.
  - Sets the label of the category trigger.
- `DetailsFormHandleEvent` now has a `detailsCategoryTrigger` case:

  If the user taps the category trigger, `DetailsFormHandleEvent` calls the `CategorySelect` system function to display categories found in the `AppInfo` block, and returns what the user selects.
- `DetailsApply` has been modified to check whether the category has been changed. If yes, it:
  - Removes all category information from the current record.
  - Sets the new category.
  - Marks the record as dirty (remembering which category the record is in).
  - Sends an `updateCategoryChanged` update code.

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 its category UI.

When the user changes the category from within the Details form, the `frmUpdateEvent` conveys 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, `FrmReturnToForm` is called before the `frmUpdateEvent` is sent (via `FrmUpdateForm`).

The order of events is important because the UI for the Edit form exists only after the `frmOpenEvent` (which was generated by `FrmReturnToForm`) has been handled. `EditViewHandleEvent` needs to have a form to update when `frmUpdateEvent` arrives.

Building and Downloading the Application

To build and download an application for use with the CodeWarrior Debugger, follow these steps:

1. **Delete any previous MemoPad applications on the device and make sure it's correctly connected with your desktop computer.**

   See Deleting the MemoPad Application for more information.

2. **Open your project (.mcp file) in CodeWarrior.**

3. **Choose Project > Remove Object Code and OK the dialog.**

4. **Choose Project > Reset Project Entry Paths.**

5. **In the Project menu, check that Disable Debugger is displayed. If Enable Debugger is displayed instead, select it to enable the debugger.**

   The debugger should be enabled if you want to debug the application you're building for the device.

6. **Press F7 to build (Make) the executable.**

   CodeWarrior creates a new .prc file and also a new .prc.psym file.

7. **Double-click the newly created .psym file.**

   CodeWarrior prompts you to enter the "shortcut .2" sequence on the device.

8. **Turn on the device and enter the sequence: ♌ .. 2, then OK the dialog on the PC screen.**

   The application is downloaded to the device. A dialog box displays the progress.

   Next, the Debugger launches the program and stops execution at the first line of the `PilotMain` function.

9. **You can now debug the application on the device just as you would debug a resident application.**

   See the CodeWarrior documentation for more information.

Preparing the Device for Testing

1. **To exercise the application in standalone mode, choose File > Exit to quit the Debugger.**

   This should reset the device.

2. **Remove the device from the cradle.**

3. **If exiting the Debugger hasn't reset the device, reset it by hand.**

   See Resetting the Device for more information.

4. **Choose the Applications silk-screened button and select the MemoPad (not Memo Pad) application in the launcher.**

Exercising the Application

With the application up and running, you can exercise the features added in this phase. Follow these steps:

Adding a New Category

1. **Select the New button on the Main form.**

   The MemoPad application displays the Edit form.

2. **Enter a few words into the text field.**

3. **Select the Details button to display the Details dialog.**

4. **Click the category name trigger.**

   The application displays the list of categories.

5. **Choose "Edit Categories...", the last entry in the list.**

   The application displays the Edit Categories dialog.

   This dialog and the Edit Categories menu are provided by the system.

Select the New button.

6. **Select the New button**

   The Edit Categories popup appears.

7. **Enter "Recreation".**

8. **Choose OK to dismiss the Edit Categories popup.**

9. **Choose the OK button to dismiss the Edit Categories dialog.**

Testing the New Category

To test the new category, follow these steps:

1. **In the Details dialog, click the category name trigger and select Recreation from the list of categories.**

2. **Click the OK button to dismiss the Details dialog.**

   The category name on the title of the Edit form is now Recreation.

3. **Click the Done button to dismiss the Edit form and display the Main form.**

   Because the category selection is "All", the new record is displayed in the list.

4. **Click the category name trigger and select Business.**

   The list is now empty.

5. **Click the category name trigger and select Recreation.**

   The record you just entered is displayed again.

Using a Table to Display the Database

Overview

In Phase 12, you replace the list in the Main form with a table.

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.
- Displaying editable or highly formatted data.

Tables give you better control over scrolling than lists. It's also more efficient to use a table when displaying records with different categories. 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 checkboxes.

One disadvantage of tables is that you can't include bitmaps in a cell. If you do, only the first "line" in the cell (one font-width high) can be selected by the user.

This phase uses only a simple one-column table to replace the list.

This phase explains how to use tables in the following sections:

Setup

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

Note: If you've just completed Phase 11, you don't need to copy `MemoPad.mcp` or the two `MemoPad.rsrc` files.

Copy these files...	From...	To...
`MemoPad.c`	`Tutorial\MemoPad 12\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 11\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 11\Src\Resource.frk`	`Apps\Working Folder\Src\Resource.frk`
`MemoPad.mcp`	`Tutorial\MemoPad 11`	`Apps\Working Folder`

Adding the Table UI to the Main Form

To add the table UI, you have to first remove the list, then add the table. To do so, follow these steps:

1. **With Constructor running, display the Main form.**
2. **Select the List that displays the list of memos and press the Backspace key (or type Ctrl-X).**

   Constructor removes the list from the form.
3. **From the Catalog, select the Table icon and drag it to the Main form.**
4. **Specify values for the table as follows:**

Object Identifier	Memo Table
Table ID	(Assigned by Constructor)
Left Origin and Top Origin	0, 18
Width, Height	160, 121

Editable	**Unchecked**
Rows	11
Column Width 1	160

5. **Save the resource file.**

Code Changes: Using a Table

The `MainViewLoadRecords` routine has been revised to load the table object with records instead of the list 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 current record is 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`.

  `MainViewLoadTable` calls `DmQueryNextInCategory` for each row of the table to find a database record in the current category for each.

  The record's index is assigned to the table row's value for column zero. This value is used later when the table row is drawn. If there are rows without records because there are too few records in the current category to fill the table, these empty rows are marked as 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 calls `MainViewDrawRecord` to draw each one of its visible table items. This new routine replaces and is similar to `MainViewListDrawItem` but it works with a list instead of a list.
- Finally, `MainViewLoadRecords` sets the first, and only, column of this table to usable so that it will draw and get user events.

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`.

Building and Downloading the Application

To build and download an application for use with the CodeWarrior Debugger, follow these steps:

1. **Delete any previous MemoPad applications on the device and make sure it's correctly connected with your desktop computer.**

   See Deleting the MemoPad Application for more information.

2. **Open your project (.mcp file) in CodeWarrior.**
3. **Choose Project > Remove Object Code and OK the dialog.**
4. **Choose Project > Reset Project Entry Paths.**
5. **In the Project menu, check that Disable Debugger is displayed. If Enable Debugger is displayed instead, select it to enable the debugger.**

   The debugger should be enabled if you want to debug the application you're building for the device.

6. **Press F7 to build (Make) the executable.**

   CodeWarrior creates a new .prc file and also a new .prc.psym file.

7. **Double-click the newly created .psym file.**

   CodeWarrior prompts you to enter the "shortcut .2" sequence on the device.

8. **Turn on the device and enter the sequence: ♌ .. 2, then OK the dialog on the PC screen.**

   The application is downloaded to the device. A dialog box displays the progress.

   Next, the Debugger launches the program and stops execution at the first line of the `PilotMain` function.

9. **You can now debug the application on the device just as you would debug a resident application.**

   See the CodeWarrior documentation for more information.

Preparing the Device for Testing

1. **To exercise the application in standalone mode, choose File > Exit to quit the Debugger.**

   This should reset the device.

2. **Remove the device from the cradle.**

3. **If exiting the Debugger hasn't reset the device, reset it by hand.**

   See Resetting the Device for more information.

4. **Choose the Applications silk-screened button and select the MemoPad (not Memo Pad) application in the launcher.**

Exercising the Application

With the application up and running, you can exercise the features added in this phase. Follow these steps:

1. **Choose the New button on the Main form.**

   The MemoPad application displays the Edit form.

2. **Enter a few words into the text field.**

3. **Choose the Done button.**

   The application returns you to the Main form, which now displays a table with a single item.

4. **Repeat steps 1-3.**

   The Main form now displays a table with two items.

By default, tables have single-line fields, so text that's longer than the screen won't wrap.

5. **Select either item.**

   The application displays the Edit form for the item.

6. **Delete the text and choose the Done button.**

   The Main form again displays a table with a single item. The application code has removed the item from the table and from the database.

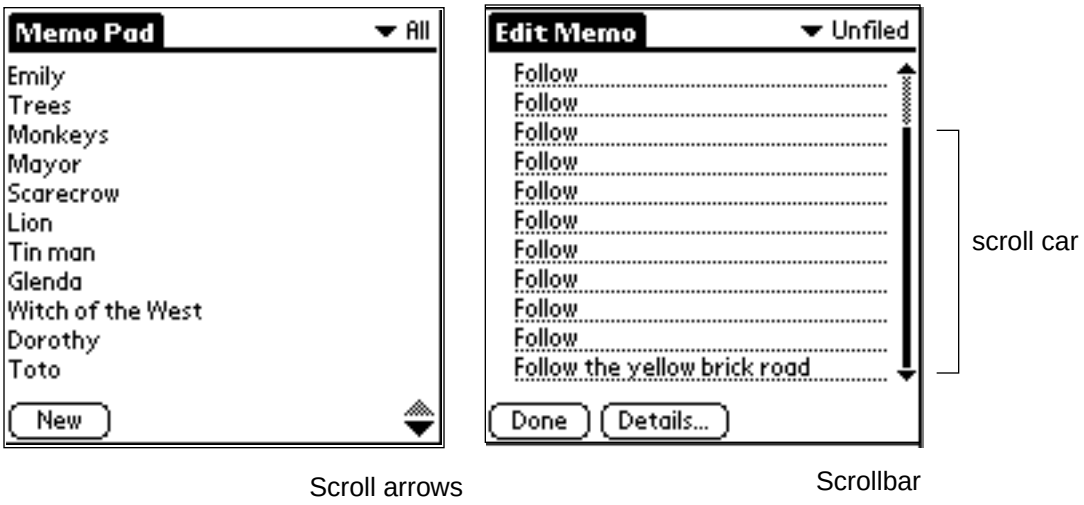
Adding Scrolling to the Main and Edit Forms

Overview

In Phase 13, you add user-controllable scrolling to the Main and Edit forms.

- For the Main form scroll arrows are added that the user can use to reposition the text.
- For the Edit form, a scrollbar will be added. As the user moves the scroll car inside the scrollbar, the text scrolls. If the user adds or removes text, the position of the scroll car in the scrollbar will be adjusted.

Note that while all the hooks are there, a large part of the scrolling behavior is actually the responsibility of the application. However, the example code for this phase illustrates clearly how to do it, so implementing it shouldn't be difficult.



Applications add scrolling capability to many data displays (lists, tables, or fields). Scrolling can involve:

- On-screen scrolling:
  - Arrow buttons (repeating or nonrepeating)
  - For 2.0 systems, a scrollbar
- Scrolling using the hard buttons on the PalmPilot device (physical scrolling).
- Both onscreen and physical scrolling.

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 PalmPilot Date Book's Week View.

Phase 13 explains how to add scrolling the following sections:

Setup

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

Note: If you've just completed Phase 12, you don't need to copy `MemoPad.mcp` or the two `MemoPad.rsrc` files.

Copy these files...	From...	To...
`MemoPad.c`	`Tutorial\MemoPad 13\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 12\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 12\Src\Resource.frk`	`Apps\Working Folder\Src\Resource.frk`
`MemoPad.mcp`	`Tutorial\MemoPad 12`	`Apps\Working Folder`

Adding Scroll Arrows to the Main Form

This section shows you how to add repeating scroll arrows to the Main form. To add the up and down scroll arrows follow these steps:

1. **With MemoPad.rsrc open in Constructor, display the Main form.**
2. **In the Catalog window, select a Repeating Button and drag it onto the form.**
3. **Set its properties as follows:**

Note: You have to remove all text from the Label and change the width before you can change the Left Origin.

Object Identifier	Scroll Up
Left and Top Origin	147, 144 **(Do this last)**
Width and Height	13, 8
Frame	Unchecked
Font	Symbol 7
Label	Click “Hex” then type 01

4. **Drag a second Repeating Button onto the form and set its properties as follows:**

Object Identifier	Scroll Down
Left and Top Origin	147, 152 **(Do this last)**
Width and Height	13 and 8
Frame	Unchecked
Font	Symbol 7
Label	Click “Hex” then type Hex 02

5. **Save the resource file.**

Code Changes: Scroll Arrow Handling for Main Form

The Main form scrolls when the user taps one of the scrolling buttons.

There are two new routines to handle button scrolling:

- `MainViewScroll` scrolls the records in the Main form's table. To do that, `MainViewScroll` simply moves n records from `TopVisibleRecord`. If there's 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 has been changed to:

- Handle the physical scroll buttons with a new case for `keyDownEvent.`
- Handle the UI (onscreen) scroll buttons with a new case for `ctlRepeatEvent.`

  The UI scroll buttons are repeating controls. As long as the user holds the pen on the button, a `ctlRepeatEvent` is queued approximately once a second. 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.

  The application processes the `ctlRepeatEvent` by scrolling the table up or down by a call to `MainViewScroll`.

Adding Scrolling to the Edit Form

This section shows you how to a add scrollbar to the Edit form.

1. **In Constructor, open MemoPad.rsrc and display the Edit form.**
2. **In the Catalog, select the Scrollbar icon and drag it onto the form, with its left border matching the right border of the edit field.**
3. **Use the following properties:**

Object Identifier	Scrollbar
Left and Top Origin	153, 18
Width	7 (leave the default)
Height	121
Value	0
Minimum Value	0
Maximum Value	0
Page Size	0

4. **Finally, display the Edit field once more and turn on Has Scrollbar.**

   As a result, the system will send more frequent `fldHeightChangedEvents` so the application can more accurately update the car position inside the scrollbar.
5. **Save the resource file.**

Code Changes: Scrollbar Handling for the Edit Form

The Edit form scrolls when the user drags the scrollbar. To achieve this behavior, the code has been changed as follows:

The new function `EditViewUpdateScrollBar` is called after a change to the text to update the position of the scrollbar car after any operation that changed the visible text.

`EditViewUpdateScrollBar` retrieves the values necessary for determining the current position of the scroll car, then calls `SclSetScrollbar` to actually set it. Note that the function allows an overlap of one line.

`EditViewUpdateScrollBar` is called by the following functions:

- `EditViewDoMenuCommand` after a go to top, go to bottom, cut, paste, or undo operation. All operations change either the amount of text or what's being displayed, so the scroll car position has to change.
- `EditViewHandleEvent` if the user entered an ASCII character.
- `EditViewRetrieveData` after that function has added text to the current text field.

Building and Downloading the Application

To build and download an application for use with the CodeWarrior Debugger, follow these steps:

1. **Delete any previous MemoPad applications on the device and make sure it's correctly connected with your desktop computer.**

   See Deleting the MemoPad Application for more information.

2. **Open your project (.mcp file) in CodeWarrior.**
3. **Choose Project > Remove Object Code and OK the dialog.**
4. **Choose Project > Reset Project Entry Paths.**

5. **In the Project menu, check that Disable Debugger is displayed. If Enable Debugger is displayed instead, select it to enable the debugger.**

   The debugger should be enabled if you want to debug the application you're building for the device.

6. **Press F7 to build (Make) the executable.**

   CodeWarrior creates a new .prc file and also a new .prc.psym file.

7. **Double-click the newly created .psym file.**

   CodeWarrior prompts you to enter the "shortcut .2" sequence on the device.

8. **Turn on the device and enter the sequence: ♌ .. 2, then OK the dialog on the PC screen.**

   The application is downloaded to the device. A dialog box displays the progress.

   Next, the Debugger launches the program and stops execution at the first line of the `PilotMain` function.

9. **You can now debug the application on the device just as you would debug a resident application.**

   See the CodeWarrior documentation for more information.

Preparing the Device for Testing

1. **To exercise the application in standalone mode, choose File > Exit to quit the Debugger.**

   This should reset the device.

2. **Remove the device from the cradle.**

3. **If exiting the Debugger hasn't reset the device, reset it by hand.**

   See Resetting the Device for more information.

4. **Choose the Applications silk-screened button and select the MemoPad (not Memo Pad) application in the launcher.**

Exercising the Application

With the application up and running, you can exercise the features added in this phase.

To test the Main form scroll buttons, follow these steps:

1. **Select the New button on the Main form.**

   The Edit form is displayed.

2. **Enter a few words into the text field.**

3. **Select the Done button to dismiss the Edit form and display the Main form.**

4. **Create at least 11 more records.**

   Hint: Use the Copy, Paste and New Page menu commands available in the Edit form (or their shortcuts) to do this more easily.

5. **Select the Done button to dismiss the Edit form and display the Main form.**

   Note that there are now scroll arrows in the bottom-right corner of the Main form.

6. **Select the scroll arrows to scroll the records that are displayed.**

   Note that pressing the up and down keys on the case has the same effect.

   To test the Edit form scroll arrows, follow these steps:

7. **Select one of the records to display an edit field.**

8. **Enter at least 20 lines of text into the text field.**

   When there are more lines than can be displayed, the scrollbar appears on the right side of the Edit form.

9. **Scroll the text fields using the scrollbar.**

10. **If you like, add or remove a large amount of text to see how the position of the scroll car changes.**

Adding System Find Support

Overview

In Phase 14, you learn how to add the system find capability. Any application that contains text data should support the system-wide find functionality of Palm OS.

Participating in this feature involves several responsibilities:

- Responding to application launch codes
- Searching your own database
- Navigating to display specific data

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 and allow any application to initiate a global search.

This phase explains how to do this in the following sections:

Setup

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

Note: If you've just completed Phase 13, you don't need to copy `MemoPad.mcp` or the two `MemoPad.rsrc` files.

Copy these files...	From...	To...
`MemoPad.c`	`Tutorial\MemoPad 14\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 13\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 13\Src\Resource.frk`	`Apps\Working Folder\Src\Resource.frk`
`MemoPad.mcp`	`Tutorial\MemoPad 13`	`Apps\Working Folder`

Adding a Find Header String

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

1. **From Constructor, open MemoPad.rsrc.**
2. **In the Constructor project window, select Strings and type Ctrl-K.**

   Constructor creates a String resource.
3. **Change the name of the resource to "Find Header Title".**
4. **Double-click on the icon.**

   Constructor displays the String Editor.
5. **Enter the text "Memos" and close the String editor**

   This string will be displayed as the heading of all found memos by the system. If you had other applications, you would choose an additional different find header title.
6. **Save and close the resource file.**

Code Changes: System Find Support

New Event Handlers in Phase 14

This phase adds event handling and other processing for the system Find functions. The standard mechanism to receive commands like this, called launch codes or launch commands, from the system is explained in this chapter.

How Launch Commands Work

Launch commands arrive at the application in the `cmd` argument to `PilotMain`. They are not events. The application is launched by the system to process these and other special commands. These system launch commands and associated parameters and flags are provided as arguments to `PilotMain`.

In Phase 14, `PilotMain` has been revised to check the `cmd` argument for a system launch command. Normally, the application is launched with the `sysAppLaunchCmdNormalLaunch` command after another application has stopped.

Note that the application may or may not be running when `PilotMain` is called with a launch command. Because of that, the application code that handles special commands **must not** use global variables set up by the application in `StartApplication` or elsewhere. After all, the `StartApplication` routine may not have been executed yet.

How a Global Find Works

There are two different launch commands the system uses to have the application perform a global find:

- `sysAppLaunchCmdFind` is the search command. An application that receives this launch command has to search its database for records that contain the specified text string (see The Search Command).
- `sysAppLaunchCmdGoTo` instructs the application to display the record found in the application's Edit view (see The Goto Command).

- `sysAppLaunchCmdSaveData` instructs the application to save its data. This is important for certain cases where the database needs to be saved before it can be updated.

The Search Command

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 private if "Hide Private Records" has been checked by the user.

The search results could include many records and be difficult to display and read if there is a large list of matches to be displayed. The system therefore allows applications to halt a search when the search results display is full, and to resume the search at the same point later.

All of this is handled by cooperation between the system and the application, as follows:

The new `Search` routine has been added to search for a text string in the application's database. It goes through these steps:

- `Search` first locates and opens the database because it can't rely on the application to be already running.
- It then calls `FindDrawHeader` to provide a string to the system Find handler, that will be displayed in the Find Results dialog as a heading for records from this application.
- 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 each record has only a single text field. If a match is found, `FindSaveMatch` is called to inform the system Find handler.
- Applications can indicate which part of the record contained the match by passing to `FindSaveMatch` values in `fieldNum` or `appCustom`. This information is passed back to the application as arguments of the `sysAppLaunchCmdGoTo` command. This can be useful for the application to position the display to a specific location within the record when it receives the `sysAppLaunchCmdGoTo` command.

- `Search` then calls `MainViewDrawRecordInBounds`, which computes the drawing bounds of the Find Results dialog. (The routine has been extracted from the `MainViewDrawRecord` routine).

  Each application should have its own version of `DrawRecordInBounds`. That way, it can display in the Find Results dialog the information from the record that is most helpful to the user. Note how each of Palm OS device's built-in PIM applications displays search results in this dialog.
- `Search` 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, `Search` close its database.

The Goto Command

Response to `sysAppLaunchCmdGoTo` is driven from `PilotMain` as follows:

- When `sysAppLaunchCmdGoTo` arrives in `PilotMain,` a check is made to see if the application is already running. The user could have initiated the search from this or any other application. If the application isn't currently running, `StartApplication` is called to get the application in a running state.
- Next, there's a call to the new routine `GoToRecord`, which responds when the user taps the "Go to" button in the Find Results dialog. The routine displays the record selected by the user. The routine does this as follows:
  - A call to `ChangeCategory` changes the current category if necessary.
  - A call to `FrmCloseAllForms` closes all open forms if the application is already running.
  - A `frmLoadEvent` is queued to start the Edit form.
  - 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 `sysAppLaunchCmdGoTo` 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).

To have the rest of the program respond appropriately, a handler has been added for each form that can receive `frmGotoEvent`. In this phase, a new case for `frmGotoEvent` has been 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.

Building and Downloading the Application

To build and download an application for use with the CodeWarrior Debugger, follow these steps:

1. **Delete any previous MemoPad applications on the device and make sure it's correctly connected with your desktop computer.**

   See Deleting the MemoPad Application for more information.

2. **Open your project (.mcp file) in CodeWarrior.**
3. **Choose Project > Remove Object Code and OK the dialog.**
4. **Choose Project > Reset Project Entry Paths.**
5. **In the Project menu, check that Disable Debugger is displayed. If Enable Debugger is displayed instead, select it to enable the debugger.**

   The debugger should be enabled if you want to debug the application you're building for the device.

6. **Press F7 to build (Make) the executable.**

   CodeWarrior creates a new .prc file and also a new .prc.psym file.

7. **Double-click the newly created .psym file.**

   CodeWarrior prompts you to enter the "shortcut .2" sequence on the device.

8. **Turn on the device and enter the sequence: ♌ .. 2, then OK the dialog on the PC screen.**

   The application is downloaded to the device. A dialog box displays the progress.

   Next, the Debugger launches the program and stops execution at the first line of the `PilotMain` function.

9. **You can now debug the application on the device just as you would debug a resident application.**

   See the CodeWarrior documentation for more information.

Preparing the Device for Testing

1. **To exercise the application in standalone mode, choose File > Exit to quit the Debugger.**

   This should reset the device.

2. **Remove the device from the cradle.**

3. **If exiting the Debugger hasn't reset the device, reset it by hand.**

   See Resetting the Device for more information.

4. **Choose the Applications silk-screened button and select the MemoPad (not Memo Pad) application in the launcher.**

Exercising the Application

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

1. **Select the New button on the Main form.**

   The Edit form is displayed.

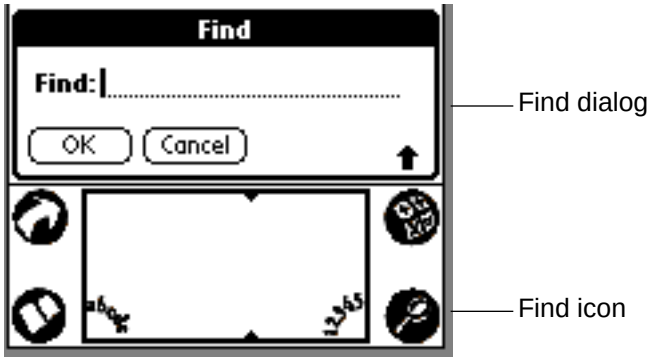
2. **Enter a few words into the text field including the word "Pilot".**

3. **Select the Done button to dismiss the Edit form and display the Main form.**

4. **Create several more records, including some that include the word "Pilot".**

5. **Select the Find icon on the display.**

   The Find dialog is displayed.



6. **Enter the word "Pilot" and tap the OK button.**

   The Find Results dialog is displayed with a list that includes several memos.

7. **Select one.**

   The Edit form containing the record is displayed. The Find string is selected.

Saving Program Settings Between Executions

Overview

In Phase 15, you add the capability to save and restore program settings to the application. The MemoPad application saves the current record and category displayed and returns the user to that display when the application is launched the next time. The information is saved in the system's preferences database.

Many applications want to save state information from one execution to the next. This includes saving current pages or viewing options, and even saving bit images of game animations.

Saving and reloading this information makes applications easier to use and contributes to the integrated, seamless appearance of the device as a whole.

To allow saving of program settings, this phase will use the Preferences database. This database is a part of the Palm OS system that all applications can use to store those options and preferences settings that are not dependent on the actual data records. For example, which view is displayed at startup isn't dependent on the content of the view.

No resource changes are made in this phase.

This phase discusses the following topics:

Setup

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

Note: If you've just completed Phase 14, you don't need to copy `MemoPad.mcp` or the two `MemoPad.rsrc` files.

Copy these files...	From...	To...
`MemoPad.c`	`Tutorial\MemoPad 15\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 14\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 14\Src\Resource.frk`	`Apps\Working Folder\Src\Resource.frk`
`MemoPad.mcp`	`Tutorial\MemoPad 14`	`Apps\Working Folder`

Code Changes: Saving Program State

The information the application has to save is currently stored in some of the application's global variables. The program therefore has been modified as follows:

- First the global variables that have to be saved were physically separated from the others in the file. This clarifies which variables are being saved and which aren't, and makes it easier to know where to put a new variable when you later add variables. A new structure, `MemoPadPreferenceType`, has been defined to contain only the variables to be saved.
- `StopApplication` has been changed 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 the Preferences database.
- The application now copies all the variables to be preserved to the `MemoPadPreferencesType` structure and then passes the structure to `PrefSetAppPreferences` to save it in the Preferences database. The system then takes this block of memory and writes it to the database.

- `StartApplication` reads the saved variables when the application starts. It calls `PrefGetAppPreferences` to read the variables 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.

Building and Downloading the Application

To build and download an application for use with the CodeWarrior Debugger, follow these steps:

1. **Delete any previous MemoPad applications on the device and make sure it's correctly connected with your desktop computer.**

   See Deleting the MemoPad Application for more information.

2. **Open your project (.mcp file) in CodeWarrior.**
3. **Choose Project > Remove Object Code and OK the dialog.**
4. **Choose Project > Reset Project Entry Paths.**
5. **In the Project menu, check that Disable Debugger is displayed. If Enable Debugger is displayed instead, select it to enable the debugger.**

   The debugger should be enabled if you want to debug the application you're building for the device.

6. **Press F7 to build (Make) the executable.**

   CodeWarrior creates a new .prc file and also a new .prc.psym file.

7. **Double-click the newly created .psym file.**

   CodeWarrior prompts you to enter the "shortcut .2" sequence on the device.

8. **Turn on the device and enter the sequence: ⍜ .. 2, then OK the dialog on the PC screen.**

   The application is downloaded to the device. A dialog box displays the progress.

   Next, the Debugger launches the program and stops execution at the first line of the `PilotMain` function.

9. **You can now debug the application on the device just as you would debug a resident application.**

   See the CodeWarrior documentation for more information.

Preparing the Device for Testing

1. **To exercise the application in standalone mode, choose File > Exit to quit the Debugger.**

   This should reset the device.

2. **Remove the device from the cradle.**

3. **If exiting the Debugger hasn't reset the device, reset it by hand.**

   See Resetting the Device for more information.

4. **Choose the Applications silk-screened button, select the MemoPad (not Memo Pad) application in the launcher.**

Exercising the Application

With the application up and running, follow these steps to verify category preservation:

1. **Select the New button on the Main form.**

   The Edit form is displayed.

2. **Enter a few words into the text field.**

3. **Select the category trigger and choose Business.**

4. **Select the Done button.**

   The Edit form is dismissed and the Main form displayed.

5. **Select the category name trigger and choose Business.**

   The category setting is now Business.

6. **Select a different applictions, for example the Address Book, on the device.**

7. **Use the launcher to return to the MemoPad application (Note that MemoPad is different from the Memo Pad, which is a standard ROM application.)**

   Notice that the category setting in the Main form is still Business.

To verify that the scroll position is preserved, follow these steps:

1. **Create several more records, at least 12.**
2. **Scroll to the bottom of the list.**
3. **Go to another application, then return to MemoPad.**

   The display should still be at the bottom of the list.

Flashy Features

Overview

In Phase 16, you add a few extra features to spice up the MemoPad application.

- The Edit form title is improved to include “n of n records” to show the position of the current record in the list of records.
- Buttons in the Edit form allow the user to change the display font.
- When a new memo is created, the Edit form now automatically sets the Graffiti Shift state.

To explain how to make these changes, this phase discusses:

Setup

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

Note: If you've just completed Phase 15, you don't need to copy `MemoPad.mcp` or the two `MemoPad.rsrc` files.

Copy these files...	From...	To...
`MemoPad.c`	`Tutorial\MemoPad 16\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 15\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 15\Src\Resource.frk`	`Apps\Working Folder\Src\Resource.frk`
`MemoPad.mcp`	`Tutorial\MemoPad 15`	`Apps\Working Folder`

Adding Resources for the Edit View Title

In this section, you change the resources to allow a variable title to appear at the top of the Edit form. The resource will contain the string `"Memo # of #"` and the program will fill in the string appropriately.

To add the string that serves as a template follow these steps:

1. **In the Constructor project window, select Strings and type Ctrl-K.**

   Constructor creates a new String resource.

2. **Set the name to "Edit View Title Template String".**

3. **Double-click on the string.**

   The String Editor appears.

4. **Enter the text**

   ```
   Memo # of #
   ```

Code Changes: Dynamic Title for the Edit Form

The `EditViewSetTitle` routine has been added to format and set the title for the Edit form. The routine sets the title to indicate the position of the record within the current category (which may be All).

- **Template for Variable Title.** A template achieves the variable title. The template contains two '#' characters.
  - The first # is replaced with the record's position.
  - The second # is replaced with the number of records in the category.

  Using a template string is especially helpful if the application needs to be localized for another language.

  The application developer can now use Constructor 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. For localization to a wide variety of languages, it may have been appropriate to use two different replacement chars, such as '#' and '%' to allow reordering of the arguments within the template string.
- **Memory.** Memory for the new title is allocated on the dynamic heap and must be freed when the form is closed by a `frmCloseEvent`, which is invoked in `EditViewHandleEvent`. Allocating memory on the dynamic heap 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 would be to retrieve the count whenever the Edit view is entered. However, if there are a large number of records in many categories this could take a long time.

  A better alternative is to call `DmNumRecordsInCategory` whenever the category selection changes, and then modify the number as the user adds and removes records. This second approach is used in the application.

- The program keeps 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.

Adding Resources for the Edit View Font Selection

This section shows you how to create font controls on the Edit form. To add two push buttons that allow users to select the font, follow these steps:

1. **Make sure the Edit form is still open.**
2. **Drag a Push Button from the catalog onto the form and set its properties as follows:.**

Object Identifier	Small Font
Push Button ID	(Assigned by Constructor)
Left Origin and Top Origin	95, 147
Width and Height	14, 12
Group ID	1
Font	Standard
Label	A

3. **Drag a second button from the catalog onto the form and set its properties as follows:**

Object Identifier	Large Font
Push Button ID	(Assigned by Constructor)
Left and Top Origin	110, 147

Width and Height	14, 12
Group ID	1
Font	Large
Label	A

The two push buttons allow the user to choose between a small and larger font. Because the buttons have the same Group ID, they are mutually exclusive.

4. **Save the resource file.**

Code Changes: Font Selection for the Edit Form

The font selection handling happens from `EditViewHandleEvent` in two ways:

- There's a new function, `EditViewChangeFont`, that handles the font change. It is called from the `ctlSelectEvent` case of `EditViewHandleEvent` and applies when either of the push buttons is tapped.

  `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 characters. It's therefore necessary to call `EditViewUpdateScrollbar` to update the scrollbar for the field.

  `EditViewChangeFont` remembers the characters in the new global variable `EditViewFont`. This allows the application to remember the font setting after the Edit view is closed and to 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.

Adding Auto-Shifting to the Edit Form

Auto-shifting in Palm OS has been enhanced to display an upper-case character under the following conditions:

- After a period or other sentence terminator (such as ? or !) and a space.
- After a period or other sentence terminator (such as ? or !) and a Return.

To turn on auto-shifting for the Edit form, follow these steps:

1. **Select the Field in the Layout Attributes Panel.**
2. **Check the Auto Shift box.**

   User input is now processed using the auto-shifting rules. If the program needs to create additional edit fields on the fly, it can set an attribute in the associated object.
3. **Save the resource file.**

Building and Downloading the Application

To build and download an application for use with the CodeWarrior Debugger, follow these steps:

1. **Delete any previous MemoPad applications on the device and make sure it's correctly connected with your desktop computer.**

   See Deleting the MemoPad Application for more information.
2. **Open your project (.mcp file) in CodeWarrior.**
3. **Choose Project > Remove Object Code and OK the dialog.**
4. **Choose Project > Reset Project Entry Paths.**
5. **In the Project menu, check that Disable Debugger is displayed. If Enable Debugger is displayed instead, select it to enable the debugger.**

   The debugger should be enabled if you want to debug the application you're building for the device.
6. **Press F7 to build (Make) the executable.**

   CodeWarrior creates a new .prc file and also a new .prc.psym file.

7. **Double-click the newly created .psym file.**

   CodeWarrior prompts you to enter the "shortcut .2" sequence on the device.

8. **Turn on the device and enter the sequence: ♌ .. 2, then OK the dialog on the PC screen.**

   The application is downloaded to the device. A dialog box displays the progress.

   Next, the Debugger launches the program and stops execution at the first line of the `PilotMain` function.

9. **You can now debug the application on the device just as you would debug a resident application.**

   See the CodeWarrior documentation for more information.

Preparing the Device for Testing

1. **To exercise the application in standalone mode, choose File > Exit to quit the Debugger.**

   This should reset the device.

2. **Remove the device from the cradle.**

3. **If exiting the Debugger hasn't reset the device, reset it by hand.**

   See Resetting the Device for more information.

4. **Choose the Applications silk-screened button and select the MemoPad (not Memo Pad) application in the launcher.**

Exercising the Application

To exercise the features added in this phase, follow these steps:

1. **Select the New button on the Main form.**

   The application displays an Edit form for a new Memo.

   Note that:

   - The title of the view now includes the number and count of memos.
   - The Graffiti Shift Indicator shows Graffiti to be in temporary up shift state. Enter a few words into the text field.

2. **Choose the Font Selection push button for the larger font.**

   Note the change in display of the field's text.

3. **Choose the Done button to dismiss the Edit form and display the Main form.**

4. **Choose a record in the list to display it in the Edit form.**

   Note that the field is displayed with the larger font and that the corresponding Font Selection button is selected.

Phase 17

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. There are a few exceptions:

- An application's database can be modified by the synchronization process, and the application must be able to handle such changes the next time it's launched.
- Applications should provide a mechanism that allows 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.

This phase helps you understand how you can make your application work well with the desktop. It discusses the following topics:

Setup

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

Note: If you've just completed Phase 16, you don't need to copy `MemoPad.mcp` or the two `MemoPad.rsrc` files.

Copy these files...	From...	To...
`MemoPad.c`	`Tutorial\MemoPad 17\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc` `MemoPadRsc.h`	`Tutorial\MemoPad 16\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 16\Src\Resource.frk`	`Apps\Working Folder\Src\Resource.frk`
`MemoPad.mcp`	`Tutorial\MemoPad 16`	`Apps\Working Folder`

Integrating With HotSync

All the changes required for this phase are in the code. There are 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. Applications should take advantage of it 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 following:

- Handling of application launch codes (`syncNotify` and `initDatabase`.)
- Maintenance of the record attribute flags in the standard database records.

If your application handles both of these areas, the system and your conduit handle the rest. To understand how conduits work, see the "Developing Palm OS Conduits" document.

Code Changes: Synchronization

This section discusses some code changes that facilitate synchronization in the following sections:

How Palm OS PIM Applications Handle Deletion

As an alternative to simply deleting records locally, the Palm OS PIM applications (Address book, Calendar, etc.) ensure that records deleted from the Palm OS device are deleted on the desktop as well. This is an important part of synchronization with the desktop.

Internally, that is, within the Palm OS device databases, this is accomplished as follows:

- The device application deletes only the body of a record and leaves a stub in the database index. This stub contains the record's unique ID and some attribute flags.
- Desktop conduits use the stub's information to delete, and possibly archive, the record from the desktop files.
- The conduit removes the stub at the end of the synchronization as part of the cleanup.

New Deletion Event Handler for the Desktop

There are three ways in which an application can handle deleted records:

- **Plain deletion**. Users delete records on the device. This is what previous phases have done.
- **Deletion with desktop support.** Applications remove records on the device but keep around a record stub.

  `EditViewSaveData` has been revised to use this new method. It now calls `DmDeleteRecord` instead of `DmRemoveRecord`. This routine sets a `deleted` flag in the record attributes and frees the rest of the record's data.

These 'deleted' stub records are ignored by routines such as `DmSeekRecordInCategory` and `DmQueryRecord`.

- **Deletion with archiving on the desktop.** Remove the records on the device, keep a stub, and allow users to archive the record to another location on the desktop. 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 deletion and archiving and let the user determine whether or not to archive a deleted record, as the Palm OS device's built-in Memo application does.

To use the archive functionality, `DmArchiveRecord` is used to set an `archive` flag in the record. If your device application supports this behavior, you must make sure the conduit supports it as well.

How Synchronization Flags Are Handled

This section explains how `MemoPad.c` handles several flags that are important for synchronization with the desktop.

The dirty Flag

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 skip unchanged records altogether.

If the `dirty` flag isn't managed appropriately, the HotSync manager has to perform a slow sync. In a slow sync, the desktop retrieves every record and compares it with its own records to determine changes since the last sync operation. In the `MemoPad.c` example for this phase, the `dirty` flag is set in `CreateRecord`, `DetailsApply`, `EditViewSelectCategory` and `EditViewSaveData`.

Note that even if the `dirty` flag is managed properly, the HotSync manager must perform a slow sync if the user synchronized with a different desktop computer during the most recent HotSync. However, this behavior is the exception, not the rule, for the Palm OS PIM applications.

The dmHdrAttrBackup Flag

If the `dmHdrAttrBackup` flag is located in the data, the database is backed up to the desktop. (More complex behavior is possible if a custom conduit exists for the application).

The simple backup can be useful if a developer doesn't want to create a conduit, but does want to back up some device data, like the high scores to a game. The desktop HotSync software can restore the database back to the Palm OS device if necessary.

For `MemoPad.c`, `StartApplication` has been revised to set the `dmHdrAttrBackup` flag when the database is created. This isn't strictly necessary because this application does have a conduit, but it's a useful precaution.

Synchronization Adaptations

Several additional code changes may be required to have your application synchronize gracefully. They are discussed in this section.

- **syncNotify launch code .** The `syncNotify` launch code must be handled in PilotMain. During the synchronization process, the HotSync manager synchronizes all records between a Palm OS device database and the desktop database. When the process is complete, the HotSync manager sends the application that created (owns) the database a `syncNotify` launch code.

  The application can then perform any work needed to handle changes to its database. This work could include:

  – Resorting the database; this is done by the Palm OS device's built-in Address Book.

  – Rescheduling all pending alarms; this is done by the Datebook.

  This MemoPad application has no such work to perform.

- **initDatabase launch code.** The `initDatabase` launch code must be handled in `PilotMain`. The `initDatabase` launch code may be sent 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.

- **Preferences.** Preferences must be checked every time the application is started in normal launch mode. In `StartApplication`, the application's preferences must be checked 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. To take this into account, this version of `StartApplication` does more thorough checking.

- **Databases.** Applications must check for a restored database every time the application is started in normal launch mode. In this version of MemoPad, `StartApplication` handles the case where the entire database has been restored from the desktop without a valid `appInfo` block.

Adding Console Commands

This phase is left blank in the Windows version of the Tutorial. For information about Console commands, see the “Palm OS Cookbook.”

Localizing for Other Countries

Overview

In this phase, you'll learn how to cleanly set up your application for localization. You'll look at the French version of the MemoPad application as an example.

The phase discusses the following topics:

Setup

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

Note: If you've just completed Phase 17, you don't need to copy `MemoPad.mcp` or the two `MemoPad.rsrc` files.

Copy these files...	From...	To...
`MemoPad.c` `Localized (folder)`	`Tutorial\MemoPad 19\Src`	`Apps\Working Folder\Src`
`AppBuildRules.h`	`Tutorial\MemoPad 19`	`Apps\Working Folder`
`MemoPad.rsrc`	`Tutorial\MemoPad 17\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 17\Src\Resource.frk`	`Apps\Working Folder\Src\Resource.frk`
`MemoPad.mcp`	`Tutorial\MemoPad 17`	`Apps\Working Folder`

Localizing Your Application

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

It's a good idea to cleanly separate the localizable and nonlocalizable portions of your program. You can do this by using separate resource files. See Chapter 1, "Designing, Testing, and Localization," of the "Palm OS Cookbook" for more information on localization.

The four Palm OS PIM applications have been localized to several languages. By following their example, you can make localization of your application a relatively straightforward process.

Localizing your application really consists of two parts:

- Translate all English-language text to the target language.

- Properly set system defaults, such as settings for dates, times, and numbers.

  This gives your application a polished look when it's used in other languages. For example, many countries use 24-hour time. Those users won't feel comfortable using 12-hour time and will appreciate it if 24-hour time is the default.

Localization Techniques

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

In your application, you have to make two kinds of changes, Changes to Resources and Code Changes.

Changes to Resources

To localize the UI resources for a different county, you have to translate all text into the desired language using Constructor (on the Macintosh you can also use ResEdit). You may have to adjust the position or size of user interface components because the translated strings may be shorter or, more likely, longer.

By convention localized Palm OS resource files are stored in a subfolder of `Src` which is named Localized. For example French version of the application's resources are in a "`Fr.XXX`" file in the `Localized` folder inside the `Src` folder.

In the ideal scenario, you have to only convert the resources and possibly reposition them, but you don't have to change the code. This is possible 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. This avoids making changes to the application code.

Code Changes

In addition, you have to change the code based on the country define. As illustrated in the Phase 19 sample code, you can create a version of `AppBuildRules.h` and include `#defines` for Country and Language to conditionally compile the application's code.

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

Palm OS 2.0 has three new functions to facilitate localization of numeric strings: `LocGetNumberSeparators`, `StrDelocalizeNumber`, and `StrLocalizeNumber`.

Example: Localization of MemoPad

In this phase of the tutorial, a local version of `AppBuildRules.h` is used to indicate during the build process that a different language is being used. To achieve this, the following pieces are needed:

- **Proper .h file**. A copy of `AppBuildRules.h` was copied from the `Incs` folder to the application folder.
- **Proper #defines**. The local copy of `AppBuildRules.h` was modified to include language and country settings:

  ```
  #define    COUNTRY    COUNTRY_FRANCE
  #define    LANGUAGE   LANGUAGE_FRENCH
  ```

  These `#defines` are used to direct the build process and to direct conditional compilation of the source code. `Incs:BuildRules.h` lists all available country choices.
- **Localized resources**. The `Localized` folder contains a modified copy of the English version of MemoPad's resources called `Fr.MemoPad.rsrc`. Several example text strings in these resources have been translated to French.

  The French resource file emits a header in the localized directory called `Fr.MemoPadRsc.h`. This file should contain the same `#defines` as the English header file "`MemoPadRsc.h`".

  `MemoPad.c` has been changed to include `Fr.MemoPadRsc.h` instead of `MemoPadRsc.h`.

Modifying the Project

Modifying the project consists of two parts:

Adding Precompiled Headers to the Project

All projects so far have used the standard system-defined `include` files. This project is using a customized `AppBuildRules.h` file and therefore has to precompile some headers itself and you have to include an `Obj` folder for the executables. Follow these steps:

1. **At top level in the Working Folder, create a new folder and name it Obj.**
2. **Open MemoPad.mcp.**

   The project window is displayed. You can now add files to the project.
3. **Choose Project > Create New Group.**
4. **Name the new group "Precompiled Headers" in the window that pops up.**
5. **You have to add three files to the new group:**
   - `Apps\Working Folder\AppBuildRules.h`
   - `Palm OS SDK\Incs\Pilot.pch`
   - `Palm OS SDK\Incs\Pilot.pch++`

   These files are used to generate precompiled headers for Palm OS UI modules.

Note: It's important that the Precompiled Headers group is first in the project. If it isn't, move it up.

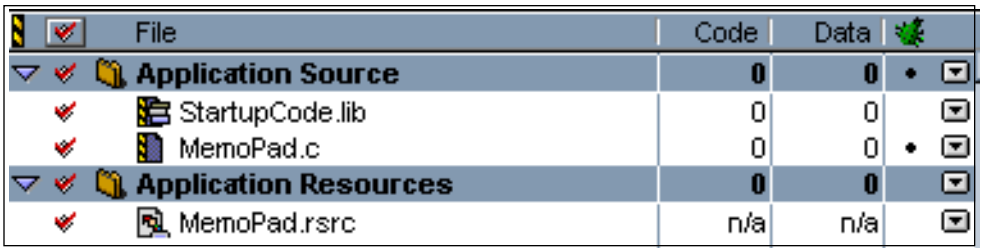
Using the Proper Resource File

The only thing missing is removing the existing resource file and adding a new one, so continue as follows:

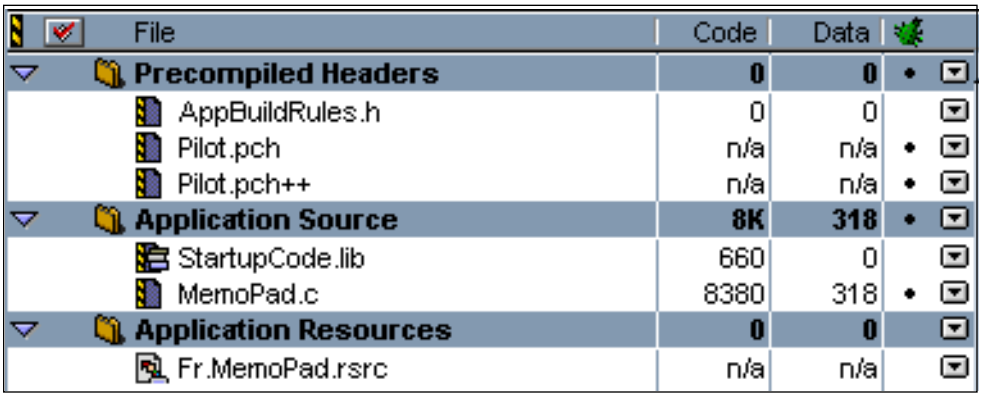
6. **Select the MemoPad.rsrc entry in the project window, then choose Project > Add Files.**
7. **Select the Localized\Fr.MemoPad.rsrc file.**

   This file contains the resources for the French MemoPad.
8. **Select the MemoPad.rsrc entry in the project window, then choose Project > Remove Selected Items.**
9. **Check that your project setup corresponds to the bottom screen shot of the figure below.**

Before (Phase 17)



After (Phase 19)



Preparing for Building the French Application

Before you can build the French version of the application, you have to check the `AppBuildRules.h` file. Follow these steps:

1. **Double-click MemoPad.mcp to open the project.**
2. **Double-click AppBuildRules.h and check that it has the proper defines:**

```
#define    COUNTRY    COUNTRY_FRANCE
#define    LANGUAGE   LANGUAGE_FRENCH
```

You're now ready to build the application.

Building and Downloading the Application

To build and download an application for use with the CodeWarrior Debugger, follow these steps:

1. **Delete any previous MemoPad applications on the device and make sure it's correctly connected with your desktop computer.**

   See Deleting the MemoPad Application for more information.
2. **Open your project (.mcp file) in CodeWarrior.**
3. **Choose Project > Remove Object Code and OK the dialog.**
4. **Choose Project > Reset Project Entry Paths.**
5. **In the Project menu, check that Disable Debugger is displayed. If Enable Debugger is displayed instead, select it to enable the debugger.**

   The debugger should be enabled if you want to debug the application you're building for the device.
6. **Press F7 to build (Make) the executable.**

   CodeWarrior creates a new .prc file and also a new .prc.psym file.
7. **Double-click the newly created .psym file.**

   CodeWarrior prompts you to enter the "shortcut .2" sequence on the device.

8. **Turn on the device and enter the sequence: ♌ .. 2, then OK the dialog on the PC screen.**

   The application is downloaded to the device. A dialog box displays the progress.

   Next, the Debugger launches the program and stops execution at the first line of the `PilotMain` function.

9. **You can now debug the application on the device just as you would debug a resident application.**

   See the CodeWarrior documentation for more information.

Preparing the Device for Testing

1. **To exercise the application in standalone mode, choose File > Exit to quit the Debugger.**

   This should reset the device.

2. **Remove the device from the cradle.**

3. **If exiting the Debugger hasn't reset the device, reset it by hand.**

   See Resetting the Device for more information.

4. **Choose the Applications silk-screened button and select the MemoPad (not Memo Pad) application in the launcher.**

Exercising the Application

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. **Select the New button on the Main form.**

   The Edit form is displayed.
3. **Note that the title and the Done and Details buttons have been localized.**
4. **Select the Details button.**

   The Details dialog is displayed.

5. **Note that the labels and buttons have been localized.**

Running the Application on PalmPilot

This phase is intentionally left blank for Windows developers. All information about running applications on the PalmPilot is already provided in earlier phases.

Phase 20: Running the Application on PalmPilot

Advanced Functionality: Application Interaction

Overview

Phase 21 illustrates both the send mail and the phone number look-up facilities. You will add a menu item to the Edit view that places the current memo in the email outbox. A second Edit view menu item prompts for a name, then looks up the corresponding phone number in the Address book and pastes the result into the memo.

The program in Phase 21 deals with more than one application:

- It illustrates how memos could be sent to the email application outbox.
- It provides phone number lookup from a memo entry to the Address book application.

Phase 21 discusses these topics:

IMPORTANT NOTE

When testing this application on the device, you must work with a PalmPilot Professional, which has the email application installed.

Note, however, that after the product was shipped, a bug was found in the built-in email application that prevents it from working correctly with the launch code sent in Phase 21. To allow you to use the email outbox feature, a revised email application has been included in the Examples folder. You can download this application when testing Phase 21. If your application needs the functionality in the revised email application, you should include it for downloading by end users.

Setup

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

Note: If you've just completed Phase 19, you don't need to copy the two `MemoPad.rsrc` files but you DO have to use the project from phase 17.

Copy these files...	From...	To...
`MemoPad.c`	`Tutorial\MemoPad 21\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 17\Src`	`Apps\Working Folder\Src`
`MemoPad.rsrc`	`Tutorial\MemoPad 17\Src\Resource.frk`	`Apps\Working Folder\Src\Resource.frk`
`MemoPad.mcp`	`Tutorial\MemoPad 17`	`Apps\Working Folder`

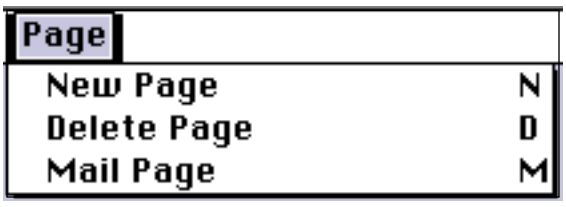
Creating New Menu Items and Alerts

This phase first steps you through creating two menus items that will become available in the Edit form, then you learn about Creating Alerts for Mail Menu.

You'll have to add two new menu items to the Edit Menu Bar:

- To the Page menu, add Mail Page.
- To the Options menu, add Phone Lookup.

The revised menus are shown below.



If you already know how to change the menus, do so and go on to the next section. Otherwise follow these steps:

1. **From Constructor, open the current version of MemoPad.rsrc.**
2. **In the Constructor project window, click on the Menu Bars icon to display all menu bars, then double-click on Edit Form.**

   The Menu Bar Editor appears. It shows the Edit Menu Bar with the Options menu displayed.
3. **Type Ctrl-K, then replace the string "untitled" with "Phone Look-up", then a Tab, and the letter "L" for the shortcut.**

   You've changed the Options menu to have a new command.
4. **Select the Page item.**

   Constructor displays the Page menu.
5. **Type Ctrl-K, then replace the string "untitled" with "Mail Page", then a Tab, and the letter M for the shortcut.**
6. **Close the Menu Bar Editor.**

   No additional changes are needed because the menu is already assigned to the Edit form.

Creating Alerts for Mail Menu

You'll have to create two alert resources to announce when mail was put into the mailbox successfully, and when the mail attempt failed. The two alerts will look as follows:

7. **Name the two alerts "Mail Sent Alert" and "Mail Missing Alert" and give them the following properties:**

Mail Sent Alert:	
Alert Type	Warning
Title	Memo Sent
Message	This memo has been sent as mail.
Button Titles: Item Text 0	OK
Mail Missing Alert:	
Alert Type	Error
Title	Mail Not Sent
Message	The Mail application is not available.
Button Titles: Item Text 0	OK

Code Changes: Phone Lookup, Mail Memo, Button Mapping

The code in Phase 21 has been changed in three ways:

Hard Button Mapping

The `StartApplication` function in Phase 21 maps the memo button (the fourth hard button) to the MemoPad application as follows:

```
PrefSetPreference( prefHard4CharAppCreator,
                   memoPadAppType);
```

It's not usually a good idea to map buttons without explicitly informing the users. This code fragment is only included to illustrate how it's done.

Mail Memo Code Changes

To support the mail menu, the code had to be changed in two places:

The `EditViewDoMenuCommand` function now has a case for the newly added Mail Memo item of the Page menu. When the user chooses this menu item, the new function `SendMemoAsMail` is invoked.

Note that the `SendMemoAsMail` function is only included as an example. In a real application, a special form should prompt the user for input such as subject, address, etc. That information could then be sent with `SendMemoAsMail.`

The function `SendMemoAsMail` performs the following actions:

- The function first creates a canned mail header using information provided by the user. The header information should match the information the email application needs. After `SendMemoAsMail` has been called, the message is ready to be

mailed, however, the user can still make changes from within the email application.

- After reading the header information, `SendMemoAsMail` retrieves the card number and database ID of the mail application, using the application's known creator type.
- `SendMemoAsMail` calls `SysAppLaunch` to add the memo as a record to the email application. To achieve that, it uses the launch command `sysAppLaunchCmdAddRecord` and the launch flag `sysAppLauchFlagSubCall`.
- If there's a problem, a `MailMissingAlertAlert` is displayed.
- If all works well, a `MailSentAlertAlert` is displayed.

Phone Lookup Code Changes

To have the system perform phone number lookup, MemoPad now has a case in `EditViewDoMenuCommand` that:

- Calls the `PhoneNumberLookup` function when the user selects the Phone Lookup menu item.
- Updates the scrollbar because phone lookup results in the addition of text, just as a paste action would.

`PhoneNumberLookup` is a system function that works as follows:

- If an item is currently selected in the passed field, `PhoneNumberLookup` looks in the address book for a match.
- Otherwise, `PhoneNumberLookup` looks for a match to the word in which the blinking cursor (insertion point) is located.
- If a match is found, `PhoneNumberLookup` replaces the selected word or the word in which the cursor is located with the phone number that's found. In effect, this replacement is a Paste action, which means the user can then select Undo.
- If no match is found, the system displays the Address Book lookup form to allow the user to select an item.

Building and Downloading the Applications

For this phase, you have to download two applications to the device, and this section therefore has two parts:

Downloading the Email Application

To download the email application to the device, follow these steps:

1. **Make sure the device is correctly connected with your desktop computer.**
2. **In the "PalmPilot Desktop" folder find "PalmPilot Install Tool" and launch it.**

   This tool is part of the end-user PalmPilot Desktop folder, not part of the SDK.
3. **Install the email application you find in the Examples folder**

Caution: If you don't install the revised email application, you can't exercise these phase. See IMPORTANT NOTE above.

Building and Downloading MemoPad

To build and download an application for use with the CodeWarrior Debugger, follow these steps:

1. **Delete any previous MemoPad applications on the device.**

   See Deleting the MemoPad Application for more information.
2. **Open your project (.mcp file) in CodeWarrior.**
3. **Choose Project > Remove Object Code and OK the dialog.**
4. **Choose Project > Reset Project Entry Paths.**
5. **In the Project menu, check that Disable Debugger is displayed. If Enable Debugger is displayed instead, select it to enable the debugger.**

   The debugger should be enabled if you want to debug the application you're building for the device.
6. **Press F7 to build (Make) the executable.**

   CodeWarrior creates a new .prc file and also a new .prc.psym file.

7. **Double-click the newly created .psym file.**

   CodeWarrior prompts you to enter the "shortcut .2" sequence on the device.

8. **Turn on the device and enter the sequence: ♌ .. 2, then OK the dialog on the PC screen.**

   The application is downloaded to the device. A dialog box displays the progress.

   Next, the Debugger launches the program and stops execution at the first line of the `PilotMain` function.

9. **You can now debug the application on the device just as you would debug a resident application.**

   See the CodeWarrior documentation for more information.

Preparing the Device for Testing

1. **To exercise the application in standalone mode, choose File > Exit to quit the Debugger.**

   This should reset the device.

2. **Remove the device from the cradle.**

3. **If exiting the Debugger hasn't reset the device, reset it by hand.**

   See Resetting the Device for more information.

4. **Choose the Applications silk-screened button, select the MemoPad (not Memo Pad) application in the launcher.**

Exercising the Application

With the application up and running, exercise the features added in this phase as follows:

1. **Tap the New button to display the Edit form and enter a name.**

2. **Leave the cursor inside or behind a word, or select some text.**

   For a successful lookup, the name must in your address book.

3. **Select the menu silk-screened button, and choose Options > Phone Lookup.**

   If you've chosen a name that existed in the Address book, the page with that name is displayed and a succession of beeps indicates success. If you've chosen a name that doesn't exist, the first page of the Address book is displayed as a phone number lookup screen and three beeps indicate failure.

   You're now ready to test the email oubox feature.

Note: To test the email outbox feature on the device, use a PalmPilot Professional and download the email application from the Examples folder.

---

4. **From within a MemoPad memo, choose Page > Mail Page.**

   A dialog will announce that the memo has been sent as mail.

5. **Choose OK in the dialog.**

6. **In the launcher, select the Mail application.**

7. **Switch views to the outbox.**

   The new memo should be visible as a message in the queue.

   You should now be ready to develop your own application. The best starting point is probably Phase 21, which has the most extensive functionality. We wish you success!

PalmPilot Resource Recipes

The resource recipes give you a place to look up the implementation of any PalmPilot resource used in the tutorial. Resources and other project elements are grouped as follows:

Catalog Resources	Button, Checkbox, Field, Form Bitmap, Gadget, Graffiti Shift Indicator, Label, List, Popup Trigger, Push Button, Repeating Button, Scrollbar, Selector Trigger, Table.
Project Resources	Forms, Menu Bars, Menus, Strings, Alerts, Bitmaps.
Project Settings	Application Icon Name, Version String, Application Icon, Generate Header File, Header File Name, Include Details in Header, Keep IDs in sync.

For more detailed information, see the CodeWarrior Constructor for Palm OS Manual and Chapter 3, “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 get ready:

1. **Open Constructor**

   A Constructor project window appears. At the top, you can instantiate Project resources, in the bottom panel, you can set Project settings.

2. **Select a Form resource template and type CtrlK.**

   A Form Editor appears. You can instantiate catalog resources by dragging them onto the Layout Appearance panel.

Catalog Resources

Button

The button resource has the following properties:

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Button ID	ID of the button. Assigned by Constructor and used to relate objects to one another (e.g., assign a menu bar to a form).
Left Origin	Number of pixels to the left edge of the container resource.
Top Origin	Number of pixels to the top of the container resource.
Width	Width of the button in pixels.
Height	Height of the button in pixels.
Usable	Sets the button to be active, visible, and a part of the interface. A nonusable object is not part of the interface and is not drawn. However, you can programmatically set nonusable to usable.
Anchor Left	Controls how the button resizes itself when its text label is changed. If the attribute is checked, then the left bound of the button is fixed; if unchecked the right bound is fixed.
Frame	If checked, the button has a frame.
Non-bold Frame	If checked, the button has a single-pixel width frame. If unchecked, the button has a bold frame.
Font	This popup sets the button's Label font.
Label	Text that appears on the button (OK, Cancel, etc.).

Checkbox

The checkbox allows users to turn settings on or off on screen. You can control the following checkbox properties:

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Checkbox ID	ID of the checkbox. Assigned by Constructor and used to relate objects to one another (e.g., assign a menu bar to a form).
Left Origin	Number of pixels to the left edge of the container resource.
Top Origin	Number of pixels to the top of the container resource.
Width	Width of the checkbox in pixels.
Height	Height of the checkbox in pixels.
Usable	Sets the checkbox to be active, visible, and a part of the interface. A nonusable object is not part of the interface and is not drawn. However, you can programmatically reset nonusable to usable.
Selected	If checked, a check mark appears in the checkbox. Otherwise, it's empty.
Group ID	Group ID for the checkbox. Grouped checkboxes are mutually exclusive. Setting the group is necessary to support this mutual behavior. Code is needed to actually enforce the behavior.
Font	Popup that sets the font for the label.
Label	Text displayed to the right of the checkbox. For a label to the left, create a Label resource and place it next to the checkbox.

Field

The field resource can display one or more lines of text.

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Field ID	ID of the field. Assigned by Constructor and used to relate objects to one another (e.g., assign a menu bar to a form).
Left Origin	Number of pixels to the left edge of the container resource.
Top Origin	Number of pixels to the top of the container resource.
Width	Width of the field in pixels.
Height	Height of the field in pixels.
Usable	Sets the checkbox to be active, visible, and a part of the interface. A nonusable object is not part of the interface and is not drawn. However, you can programmatically reset nonusable to usable.
Editable	If checked, the field is editable. Users can't enter text into a noneditable field.
Underline	If checked, the text in the field is underlined.
Single Line	If checked, the field won't scroll horizontally and won't accept tab or carriage return characters. Only a single line is displayed.
Dynamic Size	If checked, field height is expanded or contracted as characters are added or removed (dynamic sizing). Uncheck this item if **Single Line** is checked.
Left Justified	If checked, text is left justified. Supported only when **Single Line** is checked.

Max Characters	Specifies the maximum number characters allowed in the field. (32767 is the maximum.)
Font	This popup sets the font for the field.
Auto Shift	If checked, 2.0 auto-shifting is supported.
Has Scroll Bar	If checked, the field puts more frequent fldHeightChangedEvents onto the event queue.
Numeric	If checked, this is a numeric field.

Form Bitmap

Form bitmaps are used to anchor bitmaps on forms. You can set the following attributes:

Object ID	ID of the form bitmap. Assigned by Constructor and used to relate objects to one another (e.g., assign a menu bar to a form).
Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Left Origin	Number of pixels to the left edge of the container resource.
Top Origin	Number of pixels to the top of the container resource.
Bitmap Resource ID	Set this to the ID of a PICT resource to use. You can also assign an ID number, then click on Create and draw the picture in the bitmap editor that appears.
Usable	If checked, the bitmap is visible. A nonusable object is not part of the interface and is not drawn. However, you can programmatically reset nonusable to usable.

Gadget

Applications use gadget resources to indicate that a particular region of a form is handled by the application. Gadget resources allow applications to make custom UI components. You can set the following properties:

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Gadget ID	ID of the gadget. Assigned by Constructor and used to relate objects to one another (e.g., assign a menu bar to a form).
Left Origin	Number of pixels to the left edge of the container resource.
Top Origin	Number of pixels to the top of the container resource.
Width	Width of the gadget in pixels.
Height	Height of the gadget in pixels.
Usable	If checked, the gadget is visible. A nonusable object is not part of the interface and is not drawn. However, you can programmatically reset nonusable to usable.

Graffiti Shift Indicator

The Graffiti Shift Indicator specifies where to display the Graffiti Shift state indicator.

Note: By convention, the Graffiti Shift Indicator is placed in the bottom right corner of the display.

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Left Origin	Number of pixels to the left edge of the container resource.
Top Origin	Number of pixels to the top of the container resource.
Object ID	ID of the resource. Assigned by Constructor and used to relate objects to one another (e.g., assign a menu bar to a form).

States include punctuation, symbol, uppercase shift, and uppercase lock. This information is dealt with programmatically.

Label

The Label resource displays noneditable text—labels—next to buttons or checkboxes. You can change the following properties:

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Label ID	ID of the label. Assigned by Constructor and used to relate objects to one another (e.g., assign a menu bar to a form).
Left Origin	Number of pixels to the left edge of the container resource.

Top Origin	Number of pixels to the top of the container resource.
Usable	If checked, the label is visible. If unchecked, it isn't. A nonusable object is not part of the interface and is not drawn. However, you can programmatically reset nonusable to usable.
Font	This popup sets the font for the label text.
Text	String displayed in the label.

List

The list resource has the following properties:

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
List ID	ID of the list. Assigned by Constructor and used to relate objects to one another (e.g., assign a menu bar to a form).
Left Origin	Number of pixels to the left edge of the container resource.
Top Origin	Number of pixels to the top of the container resource.
Width	The width of a list item in pixels.
Usable	Checked sets the form to be visible and unchecked to be not visible. A nonusable object is not part of the interface and is not drawn. However, you can programmatically reset nonusable to usable.
Font	This popup sets the font.

Visible Items	Height of list (in number of visible items). If you enter zero, all items are considered visible.
List 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.

Popup Trigger

When a user selects a popup trigger, the system displays a list and the user can then select an item from that list.

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Popup ID	ID of the popup trigger. Assigned by Constructor and used to relate objects to one another (e.g., assign a menu bar to a form).
Left Origin	Number of pixels to the left edge of the container resource.
Top Origin	Number of pixels to the top of the container resource.
Width	Width of the trigger in pixels.
Height	Height of the trigger in pixels.
Usable	If checked, the trigger is active, visible, and a part of the interface. A nonusable object is not part of the interface and is not drawn. However, you can programmatically reset nonusable to usable.
Anchor Left	Checked fixes the left bound of the trigger when the size of the label changes.
Font	This popup sets the font for the popup trigger. Use Symbol 7 and Hex 01/Hex/02 for up/down arrows.

Label	The caption that appears on the trigger.
List ID	ID of the list object that should appear when the user taps the trigger.

Push Button

Push buttons usually indicate exclusive options. You can change the following attributes:

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Button ID	ID of the push button. Assigned by Constructor and used to relate objects to one another (e.g., assign a menu bar to a form).
Left Origin	Number of pixels to the left edge of the container resource.
Top Origin	Number of pixels to the top of the container resource.
Width	Width of the push button in pixels.
Height	Height of the push button in pixels.
Usable	If checked, the push button is active, visible, and a part of the interface. A nonusable object is not part of the interface and is not drawn. However, you can programmatically reset nonusable to usable.
Group ID	Group number for the push button. Grouped push buttons are mutually exclusive. This behavior has to be enforced programmatically.
Font	This popup sets the font.
Label	The text that appears on the push button.

Repeating Button

Repeating buttons are identical to buttons except they repeatedly send events while they are pushed.

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Button ID	ID of the push button. Assigned by Constructor and used to relate objects to one another (e.g., assign a menu bar to a form).
Left Origin	Number of pixels to the left edge of the container resource.
Top Origin	Number of pixels to the top of the container resource.
Width	Width of the repeat button in pixels.
Height	Height of the repeat button in pixels.
Usable	If checked, the repeat button is set to be active, visible, and a part of the interface. A nonusable object is not part of the interface and is not drawn. However, you can programmatically reset nonusable to usable.
Anchor Left	If checked, the left bound of the button is fixed when the size of the label changes.
Frame	Checked gives the repeat button a frame.
Non-bold frame	Checked gives the repeat button a plain frame. Unchecked creates a bold frame.
Font	This popup sets the font for the label.
Label	Text that appears on the repeat button.

Scrollbar

Scrollbars are independent resources that you need to place on the form in the appropriate position next to a field or table. The scrollbar has to be updated programmatically. You can change the following properties:

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Scrollbar ID	ID of the scrollbar. Assigned by Constructor and used to relate objects to one another (e.g., assign a menu bar to a form).
Left Origin	Number of pixels to the left edge of the container resource.
Top Origin	Number of pixels to the top of the container resource.
Width	Width of the scrollbar in pixels. Default and recommended value is 7.
Height	Height of the scrollbar in pixels.
Usable	If checked, the scrollbar is set to be active, visible, and a part of the interface. A nonusable object is not part of the interface and is not drawn. However, you can programmatically reset nonusable to usable.
Value	Current position of the scroll car inside the scrollbar.
Minimum Value	Position of scroll car inside the scrollbar when text is scrolled to top. Should be 0.

Maximum Value	Position of scroll car when text is scrolled to bottom. Should be Total lines - page size + overlap.
Page Size	Number of lines that fit into the field.

Selector Trigger

A selector trigger indicates to the user that a dialog will appear to edit the contents of the selector.

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Selector Trigger ID	ID of the selector trigger. Assigned by Constructor and used to relate objects to one another (e.g., assign a menu bar to a form).
Left Origin	Number of pixels to the left edge of the container resource.
Top Origin	Number of pixels to the top of the container resource.
Width	Width of the repeat button in pixels.
Height	Height of the repeat button in pixels.
Usable	If checked, the repeat button is set to be active, visible, and a part of the interface. A nonusable object is not part of the interface and is not drawn. However, you can programmatically reset nonusable to usable.
Anchor Left	If checked, the left bound of the selector trigger is fixed when the size of the label changes.

Font	This number sets the selector trigger's Label font.
Label	Text that appears on the selector trigger.

Table

The table resource lets you change the following fields:

Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
Table ID	ID of the table. Assigned by Constructor and used to relate objects to one another (e.g., assign a menu bar to a form).
Left Origin	Number of pixels to the left edge of the container resource.
Top Origin	Number of pixels to the top of the container resource.
Width	Width of the table in pixels.
Height	Height of the table in pixels.
Editable	If checked, the table is editable. A noneditable table won't accept text.
Rows	The number of rows in the table.
Columns	The number of columns in the table. You add columns by creating additional copies of "Column Width 1". The system then updates this column number.
Column Width 1	Select **Column Width 1**, then type Ctrl-K to add a column to the table.

Project Resources

Forms

A form acts as a container for other resources. You can change the following form properties:

Left Origin	Number of pixels to the left edge of the display.
Top Origin	Number of pixels to the top of the display.
Width	Width of the form in pixels.
Height	Height of the form in pixels.
Usable	Not currently supported.
Modal	If checked, the system 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	If checked, the system saves pixels behind the form and restores them when the form goes away. Generally, this option is used with dialog boxes.
Form ID	Each form is assigned its own ID number by Constructor.
Help ID	Developers can enter an ID number of a string that's displayed when the user taps the "i" icon here. The system adds the icon to the form when you assign a value to this property.
menu bar ID	Developers can enter an ID number of a menu bar for this form here.
Default Button ID	Developers can specify a button number here. The system assumes the default button is selected if the user switches to another application, forcing the form to go away.

Form Title	Title of the form. The title must be one line; it uses about 13 pixels of the top of the form. Use titles for dialogs, menu bars for views.
Palm OS Version	Version of the device for which this form is created.

Menu Bars

Menu Bars provides a template for a menu bar and act as a starting point for menu creation.

To...	Do this...
Create a menu bar	Select menu bars and type Ctrl-K.
Edit the menu bar	Select the menu bar instance and double-click.
Add menus	In the menu editor, type Ctrl-M. Interactively change the name and, optionally, position of the menu.
Add menu items	In the menu editor, select a menu and type Ctrl-D.
Add shortcuts	In the Menu Editor, select a menu item and type Ctrl-I to display the Property inspector. Enter the shortcut into the Shortcuts area.
Associate the menu with a form	Enter the menu bar ID into the menu bar ID field of the form's Layout Properties. When an application is running, the menu only becomes visible when the user selects a shortcut or selects the menu icon while the form is displayed.

Menus

It's easy to add menus as you're creating the menu bar. In that case, Constructor creates the instances of Menus for you. See Menu Bars for information on menu creation.

Strings

Create instances of strings to use as labels.

To create a string:

1. **Select Strings and type Ctrl-K.**

   A new string instance appears.

2. **Double click the string instance.**

   The String Editor appears.

3. **Enter the text and close the editor.**

   You can now assign the string to other resources.

Alerts

Alerts allow you to display a warning dialog to the user. To create an alert, select Alerts and type Ctrl-K. Double-click on the alert to enter this information:

Alert Type	Determines which of four possible icons is displayed in the alert and the sounds that it plays when the alert is drawn. Choose from Information, Confirmation, Warning, Error.
Help ID	ID of a String resource that is the help text for the Alert dialog box. If you provide a value, the system displays an "i" in the top-right corner of the Alert box.
Default Button	Developers can specify a button number here. The system assumes the default button is selected if the user switches to another application, forcing the form to go away.

Title	Title displayed on the Alert dialog.
Message	The message displayed on the Alert dialog. May contain ^1, ^2, ^3 as substitution variables for use in conjunction with `FrmCustomAlert`.
Button Titles	Opens to a list of button names. For each button, you can change the text. To add a button, select Item Text 0, and type Ctrl-K.

NOTE: The Alert dialog is always displayed programmatically. It therefore doesn't make sense to add it to the form.

Icons

This resource lets you create an application icon that will later be used by the launcher to allow users to display your application.

Bitmaps

This resource lets you associate Macintosh PICT formats with your interface.

Project Settings

This section only gives a brief overview of the project settings. For more detailed information, see the CodeWarrior Constructor documentation.

Generate App Resources

Should be checked.

Application Icon Name

The application icon name is the name that appears in the launcher together with the Application icon. Choose a string that's short for easy access.

Version String

The version string allows you to enter a version to your application.

Application Icon

The application icon is the image that appears in the launcher. Only the top 20 lines of the 32 available are used. This is a Macintosh resource, not a Palm OS resource.

Generate Header File

Instructs Constructor to generate a header file for your project.

Header File Name

Name of header file Constructor generates.

Include Details in Header

Includes resource details as comments in the generated header file.

Keep IDs in sync

Updates IDs in header file when Constructor IDs are updated.

Appendix A: PalmPilot Resource Recipes

Appendix B

Tutorial Troubleshooting

This appendix lists some frequently encountered problems and their solutions.

- **When you try to create resources, they already exist in the .rsrc file.**

  Look at the instructions once again: You always copy the source files from the current phase but the resource file from the previous phase.

- **Constructor message: "Couldn't generate header, perhaps the file is open in CodeWarrior" when saving a Constructor file.**

  This may happen when another file in the same project is open or locked. Note that the project itself doesn't have to be locked.

- **Build problems.**

  Make sure that the included `.h` file in your code files has the same name as the header file generated by CodeWarrior.

- **You build your project after making changes but the changes don't seem to take into effect. For example, you replace the English with the French resource file but Simulator displays the English version.**

  You may still have the English object code in your `Obj` folder. Choose Project > Remove Binaries; then compile and run again.

- **Error: MemoPadRsc.h couldn't be found.**

  The tutorial assumes that you work through the phases consecutively. Each time you save a resource file, Constructor generates a `MemoPadRsc.h` file. I've you've started with one of the later phases, you can either copy the file from the `Tutorial` folder or open the `.rsrc` file once in Constructor, then save it to have Constructor generate the corresponding header.

- **When you're trying to download your application to the device, you can't connect.**

Make sure that HotSync isn't running. HotSync and the Debugger can't use the desktop computer's serial port simultaneously.

- **You've downloaded your application to the device but no source code is available for debugging.**

  Make sure the project file is set up to allow source-level debugging. In the project file, a small dot must appear to the right of AppSources and next to the filename in the column that has a small green bug in the header. If there is no dot, click in the "bug" column to turn it on.

- **Under Windows Explorer, you can't see the extension to .c and .rsrc files.**

  Make sure that the view option "Show DOS file extensions" is turned on.

- **Code is badly formatted.**

  In CodeWarrior, choose Edit > Preferences > Fonts & Tabs, then set Tab Size to 2.

Appendix B: Tutorial Troubleshooting

Appendix B: Tutorial Troubleshooting

Appendix B: Tutorial Troubleshooting

Appendix B: Tutorial Troubleshooting

Appendix B: Tutorial Troubleshooting

Appendix B: Tutorial Troubleshooting

Appendix B: Tutorial Troubleshooting

Appendix B: Tutorial Troubleshooting

Index

Symbols

A

B

C

D

E

F

T

U

V