

Metrowerks Tutorial:
CodeWarrior[®] Development Studio
for Palm OS[®]

Revised: <04/18/03>

©Copyright 1993-2001. Metrowerks Corp. ALL RIGHTS RESERVED.

Metrowerks and the Metrowerks logo are trademarks or registered trademarks of Metrowerks Corp. and/or its subsidiaries in the United States and other countries.

CodeWarrior, PowerPlant, Metrowerks University, CodeWarrior Constructor, Geekware, and Discover programming are trademarks and/or registered trademarks of Metrowerks Corp. in the United States and other countries.

Graffiti, HotSync, Palm OS and Palm Modem are registered trademarks, and Palm III, Palm IIIx, Palm V, Palm VII, Palm, More connected., Simply Palm, the Palm, Inc. logo, Palm III logo, Palm IIIx logo, Palm V logo, Palm VII logo, and HotSync logo are trademarks of Palm, Inc. or its subsidiaries. All other trademarks are the property of their respective owners and are hereby recognized.

Documentation stored on the compact disk(s) may be printed by Licensee solely 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 Corp.

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

If your current licensing key has restrictions, you may experience difficulty using CodeWarrior, depending on those restrictions. Contact Metrowerks for more information and to obtain a full licensing key by sending email to sales@metrowerks.com or by calling 800-377-5416 or +1-512-997-4700 (international).

How to Contact Metrowerks:

U.S.A. and International	Metrowerks Corporation 9801 Metric Blvd., Suite #100 Austin, TX 78758 U.S.A.
World Wide Web	http://www.metrowerks.com
Technical Support	mailto:support@metrowerks.com USA: 1-800-377-5416 Intl.: +1-512-997-4700

Table of Contents

1 Introduction	7
What's New	7
Purpose	9
The Requirements	9
Choosing a Resource Editor	9
Resetting the Device	10
Soft Reset	10
No-notify Reset (Warm Reset)	11
Hard Reset	11
How to Get the Most out of this Tutorial	11
Where to Get More Information	12
CodeWarrior Documentation	12
Palm Documentation	13
Sample Code	13
Knowledge Base	13
Newsgroups and Forums	14
 2 Setting Up the Project	 15
Create the Project	15
Modify the Stationery	20
Compile, Debug, and Run	21
Take a Look at the Code	25
 3 A Form and an Alert	 27
Set up the Resource File	27
Create a Form	29
Create an Alert	30
Switching Between Resources.. . . .	32
Compile and Run	32
Take a Look at the Code	34
 4 Two Buttons and Another Form	 36
Create a Button on the Main Form	36
Create a New Form	37
Put a Button on the Create Form	38
Compile and Run	38

Take a Look at the Code	39
5 An About Form, Menus and Menu Bars	40
Create an About Form	40
Create a Menu and a Menu Bar	42
Compile and Run	42
Take a Look at the Code	44
6 Labels and a Help String	45
Add Labels to the About Form	45
Add a Help String to the Form	46
Compile and Run	47
Take a Look at the Code	48
7 Adding Bitmaps to the About Form	49
Learn About Bitmap Families	49
Add Bitmaps to the About Form	50
Compile and Run	52
Take a Look at the Code	53
8 Add a Field to a Form	54
Add a Field and a GSI	54
Create an Edit Menu	55
Compile and Run	56
Take a Look at the Code	57
9 A List, a Button, and an Alert	59
Create and Populate a List	59
Add a Button and an Alert	60
Compile and Run	61
Take a Look at the Code	62
10 Popup Trigger and Saving to a Database	64
Create a Popup Trigger.	64
Add a Button and Two Alerts	66
Compile and Run	67
Take a Look at the Code	68
11 Adding a View Form	72
Create a Form with Push Buttons	72
Add a Field and Two Buttons	73
Compile and Run	74
Take a Look at the Code	75

12 A Form for Completion and Deletion	77
Create a Form, Buttons, and an Alert	77
Add Several Labels and Checkboxes	78
Compile and Run	80
Take a Look at the Code	80
13 Selector Triggers for Reminders	83
Put Selector Triggers on the Complete Form.	83
Create a Form to be Triggered by the Selector	84
Compile and Run	86
Take a Look at the Code	86
14 Wrapping Up the Application	88
Copy a Bitmap From an External Editor	88
Create a Large Application Icon	89
Create a Small Application Icon	90
Compile and Run	91
Take a Look at the Code	91
Conclusion	92

Introduction

Welcome to the CodeWarrior for Palm OS Tutorial. In this section you will see information on:

- What's New
- The Purpose
- The Requirements
- Choosing a Resource Editor
- Resetting the Device
- How to Get the Most out of this Tutorial
- Where to Get More Information

CodeWarrior Development Studio for Palm OS Platform version 9.0 is a fully integrated development environment for the Palm OS platform. It comes with the 5.0 SDK (the latest version of the SDK available at the time this manual went to print). This new SDK allows you to develop for Palm OS 5, the first Palm OS that supports the next generation ARM-compliant processors. However, just like previous SDKs, you can develop for any device that runs any version of the Palm OS from the original 1.0 up to the recent 5.0. This is done through the marriage of C or C++ code and the resources that make up the user interface (UI). Resources include anything the user can interact with – forms, menus, buttons, etc. In this section, you will see a list of the various UI elements that can be added to a Palm OS application.

Regardless of how well you already know C or C++, Palm development has its own learning curve. This is due to the fact that Palm, Inc., has created a comprehensive Application Programming Interface (API) for development for this OS. As a result, a new Palm OS developer must learn about creating and using the various resources properly as well as learn about the Palm OS API that is used to control the resources.

What's New

There are number of new additions included with this latest release of the CodeWarrior for Palm OS development tools from the previous version. A complete list of what's new in this release can be found in the developer notes and release notes of your CodeWarrior installation directory. A few of the most important additions for the purposes of this tutorial are Palm OS 5, the new resource editor, and the new Palm OS linker.

Palm OS 5

The biggest change in Palm OS 5 is the new hardware platform. Palm OS 5 devices use ARM-compliant processors whereas older devices used Motorola Dragonball 68K processors. The new hardware provides substantial improvements in speed and efficiency. Thus, developers can use this new platform to create advanced software

applications while still keeping in stride with the elements of the Palm OS philosophy: ease of use, power efficiency, and low cost.

Although, this is an entirely new hardware platform, existing Palm OS applications can still be run without modification or changing source code. This is accomplished by the Palm Application Compatibility Environment, or PACE, which is a compatibility layer that all 68K applications must run through. PACE does not emulate 68K hardware or the older OS. Instead, it interprets and translates 68K instructions to work on the new hardware. The operating system, PACE itself, and all API function calls are written in native ARM code. Applications will still work as before, but will run much faster.

Palm OS 5 is actually an introductory OS for the new ARM platform. Most Palm OS 5 applications do not need native ARM code and will not benefit from using native ARM code. Palm OS 5 itself runs as ARM code, so all API functions run at the full speed of the ARM processor. However, for performance reasons, some algorithms may need to be written in native ARM. For this reason, the OS also lets you call native code through a special back door. ARMlets are small sections of native ARM code that can be embedded within 68K applications. Although we offer tools for creating ARMlets in this release, this tutorial will not cover the topic. Please refer to the ARMlet tools documentation if you are interested in learning how to create ARMlets.

The Palm OS 5 Simulator is the primary tool used for testing and debugging Palm OS 5 applications. The Simulator implements the Palm OS platform on Windows. It is very similar to the Palm OS Emulator in the way it looks, runs applications, and how it connects to external debuggers. However, these two tools are very different in terms of implementation and capabilities. It's a good idea to use both the Emulator and the Simulator in order to utilize the testing features unique to each tool. However, we will only need to use the Emulator in this tutorial.

Resource Editor

CodeWarrior for Palm OS, version 9 offers a new tool for Palm OS resource editing in addition to Constructor for Palm OS. PilRC Designer is a graphical Palm OS resource editor that offers developers an alternative to Constructor for Palm OS, which is included in the Palm OS SDK and was used as the standard resource editing tool in previous versions of CodeWarrior for Palm OS. We will be using PilRC Designer in order to create resources instead of Constructor throughout this tutorial.

Palm OS Linker

The previous version of the CodeWarrior for Palm OS toolset relied on the Macintosh 68K tools. The resources were created in Mac OS format, a 68K Mac OS application is produced as an intermediate form, and the PalmRez post-linker supplied by Palm is used to create the final executable (PRC) file. CodeWarrior for Palm OS V9 includes a new linker that removes the dependency on the PalmRez post linker by working directly with Palm OS formats throughout the build process and integrating Mac OS formats only for importing older projects. Thus, in the new build flow, there is no need for the PalmRez post linker panel that was found in previous versions of the tools. This panel has been replaced by others that will be covered later on within this tutorial. However, the PalmRez post linker panel will still appear in projects that still use the old build flow.

Debugger

CodeWarrior for Palm OS, V9 includes a completely new and revamped debugger that adds greater usability and flexibility to Palm OS development and testing. Along with the new debugger comes a new settings panel. The global PalmDebugger settings panel has been moved to the Palm OS Debugging panel, which is located in the Target settings window of the project. The Palm OS Debugging settings panel will be covered later.

IDE 5

CodeWarrior for Palm OS V9 also includes a new IDE that offers many new features that are not found in older versions of the IDE. The following outlines some of the major features that are included: code completion similar to Visual C++, docking windows, tabbed windows, and workspace support that allows users to save configuration of projects and windows that can be reloaded at a later time. Please refer to the [IDE 5.0 User Guide](#) for more information on this IDE.

The Purpose

The primary purpose of this tutorial is to show a new Palm OS developer how to create resources properly, and then discuss the code that is required to make them work. This tutorial can also be useful for current CodeWarrior for Palm OS developers who are upgrading to V9 and who would like to familiarize themselves with the new features and build flow within this release. This tutorial is not intended to teach the Palm OS API, though some calls will be discussed in the process of discussing the code requirements. You will not write any code, instead you will copy it from a folder that contains the updated source files. You will go through the steps of creating the resources, looking at the code, and debugging the application.

This tutorial is broken down into chapters that cover a certain type of resource(s). At the end of each chapter you will have an application that is fully functional and has more features than it did at the end of the previous chapter. It is set up so that you can work from beginning to end to learn progressively more difficult concepts. You can also choose to jump to a specific chapter and begin there by simply copying the files from the Tutorial folder for the chapter you are about to begin.

The Requirements

If you are reading this document, you most likely have CodeWarrior for Palm OS Platform version 9.0. This is the first thing that is required to go through this tutorial. Install it on your machine using the installer. Then you need to have a device that runs the Palm OS or a ROM file from one of the Palm OS device vendors. Note that the Palm OS Emulator and the Palm OS Simulator both require ROM files in order to work. However, each tool uses a different type of ROM file.

Now that you have the hardware and software necessary, begin with a good understanding of either C or C++. You must be proficient in your desktop development environment. You should have used a Palm OS device before, and be familiar with the way they act and look.

Choosing a Resource Editor

If you read the introduction, you understand that you will be learning to create resources, and you will see the code that must be created to make these resources

function properly. These resources can be developed in a number of applications, so when you create a new application, you'll want to decide which is right for your project. The resource editing options included in this release of the product are Constructor for Palm OS, PilRC Designer, and Rez. This is the first release that includes two different graphical resource editors, Constructor for Palm OS and PilRC Designer. In the following paragraphs, you'll see the strong and weak points of the various tools so you can make an educated choice.

Constructor for Palm OS is a free tool provided and maintained by Palm, as part of the SDK. It is a graphical WYSIWYG (what you see is what you get) tool, so you are able to see a representation of the form you are developing before you put it on an actual Palm device. Since this application was originally a port of the Constructor for Macintosh product, some Windows users find it non-intuitive and hard to use. Constructor creates files with a .rsrc extension.

PilRC Designer is a visual resource editor for the Palm OS that also displays a representation of the forms and controls that will be included in the application before it is placed on the actual device. PilRC Designer operates on the following resource file types: the text-based PilRC format that is understood by the PilRC resource compiler discussed below and the .rsrc file format that is understood by Constructor for Palm OS.

PilRC is a free tool available under the GNU Public License (GPL), and is maintained by Aaron Ardiri (a longtime member of the Palm OS developer community, but unaffiliated with Metrowerks or Palm). This tool is a text-based resource editor, so you can't see a mockup of the form until you run the application on a device or the emulator. However, because it is text-based, the developer has a significant amount of flexibility that isn't available with Constructor. PilRC creates files with a .rcp extension. A PilRC compiler plugin, developed by Callope Enterprises, Inc., is included with CodeWarrior for Palm OS for compiling .rcp files into Palm OS resources.

Rez is also a text-based tool, but it is based on the Apple Rez format for Macintosh resources. People who are familiar with the Rez format from programming for the Mac OS may find this tool useful. If not, there is not very much Palm OS-specific documentation for using Rez, so if you would like to learn a text-based resource format for the Palm OS, you are better off to stick with PilRC. The Rez compiler expects text files with a .r extension.

While these four resource editors all have their strong and weak points, you are not limited to using only one of these tools. You can have .rsrc, .rcp, and .r files all in the same project. So, you are free to use the best tool for the job. In this tutorial, we will use PilRC Designer. The previous tutorial used Constructor for Palm OS. Both tools tend to be fairly straightforward for new users getting into Palm OS development, but PilRC Designer offers a little more flexibility since it uses text-based resource files.

Resetting the Device

Periodically during the development cycle, certain actions will require you to reset your device. Most often this will happen if the debugger fails in its connection with the device, or if there is a problem with your application that forces your device into an infinitely resetting loop. There are ways of resetting the device to regain functionality of the device that don't include sending it back to the manufacturer. The three main resets are

- Soft Reset

- No-notify Reset (Warm Reset)
- Hard Reset

Soft Reset

This is the type of reset you will have to do if your debugger download fails. When you start the download the serial port is opened for the debugger, and then if that fails, the serial port is not closed. This type of reset will close the serial port on the device and clear the dynamic heap without disturbing the storage heap. This means the data on your device will be untouched, but there is a clean stack and global variable area, the drivers are restarted and the serial port is reset. Perform the reset by inserting the end of your stylus (some styli can have the top or the tip unscrewed to reveal a tool for resetting the device) or an opened paperclip into a small hole on the back of the device, which is labeled "Reset". All applications on the device will receive a `sysAppLaunchCmdReset` launch code from the OS with this reset.

No-notify Reset (Warm Reset)

If an application is misbehaving, it can cause the device to get stuck in a soft reset. When the reset launch code is sent out to all applications, if one doesn't handle the code properly, the device will continue to soft reset continually. To get out of this state, you need to do a no-notify reset, which is just like a soft reset, but the `sysAppLaunchCmdReset` code won't be sent out to the applications and no system patches will be loaded. Perform this reset by pressing and holding the up button on the device while poking the reset button on the back of the device. Once the device has reset properly, delete the application from memory to avoid further problems.

Hard Reset

If a soft reset and a no-notify reset are not enough to get your device back to a working state, the last course of action is a hard reset. A hard reset will clear the storage heap as well as all the actions of a soft reset. This has the effect that all the user data and applications are removed from the device, rendering it back to the state it was in when you first took it out of its box. Since hard resets are occasionally necessary, it only makes sense to sync your device with the HotSync Manager before beginning your development. To perform a hard reset, press and hold the power button on the device, while poking the reset button on the back of the device. Hold the power button down until the Palm OS logo appears on the screen. If you have done this correctly, you will get a confirmation for deleting all the data. After selecting yes (by pressing the up button), the device will complete its reset.

How to Get the Most out of this Tutorial

For a developer new to the Palm OS Platform, start with Chapter 1 and proceed to the end of this manual. Each chapter builds on what has been demonstrated in the previous chapters, so if you've skipped a chapter or more, you may get lost. In each chapter, modify your resource file according to the directions provided for the resource editor you have chosen. Then copy over the updated source code to handle the new resources, and go through the discussion of the code that was added. You should look up some of the API calls in the [Palm OS Reference](#) manual. This will give you a feel for the way the API calls are documented and the way they are used in the code. Debug the project, and step through the code until you understand what the code is doing and why.

The code you will need for each chapter is found in the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand folder. Don't modify this folder. Instead, work in a separate

folder, and copy over the necessary files to a working directory (like a CodeWarrior Manuals\MW Tutorial for Palm OS\MyErrand folder) to modify them. In each of the Chapter folders is the information you will need to begin the specified chapter. Most of these folders contain a project file, Src folder and Rsc folder. If you work the tutorial in the order it is presented you will be updating the resource file and the project file as you progress. Therefore you will not need the project file or the information in the Rsc folder. Instead, you will only take the code from the Src folder and replace the sources in your working directory to be ready to work through the next chapter. If you have a chapter that doesn't seem to work for you, or if you are not working the tutorial chronologically, copy the entire contents of the chapter to your MyErrand folder, and then begin working through the chapter.

Where to Get More Information

There are lots of resources that are available to you on the web and the CodeWarrior for Palm OS Platform version 9.0 CD. A few of them are described in this section:

- CodeWarrior Documentation
- Palm Documentation
- PilRC Documentation
- Sample Code
- Knowledge Base
- Newsgroups and Forums

CodeWarrior Documentation

Documentation for using the CodeWarrior toolset is found in the CodeWarrior Manuals\HTML\ folder of your installation except for Targeting Palm OS Platform, which is found in CW for Palm OS Support\Documentation\Targeting Palm OS. The HTML manuals are also accessible through the IDE Help menu via the Online Manuals menu option. Most of these manuals are written for the collective set of tools that Metrowerks produces, including the desktop, embedded, and games tools. A brief synopsis of all the CodeWarrior manuals in your installation is below.

<i>C Compilers Reference</i>	This manual discusses the pragmas and symbols used in Metrowerks' implementation of C and C++. It also provides information about the compiler settings panels.
<i>IDE 5.0 User Guide</i>	This manual tells you everything you could ever want to know about the CodeWarrior project file, the editor, and how to use the various features of the debugger.
<i>MSL C Reference</i>	This manual contains the description of all the header files, extensions, and libraries included in the Metrowerks C implementation.
<i>MSL C++ Reference</i>	Here you find out all about the C++ implementation Metrowerks has made.
<i>Extending the CodeWarrior IDE</i>	This manual provides material on extending the standard features of the CodeWarrior IDE through scripts and enhancement plug-ins.
<i>COM API Reference</i>	This manual discusses how to use the COM (Common Object Model) API's to control the CodeWarrior IDE.
<i>Targeting Palm OS Platform</i>	This manual discusses the features of the product that are specific to Palm OS development. Here you can find

	information on the settings in the various panels, the differences between project types, debugging and troubleshooting information, and more.
<i>PilRC Designer Help</i>	This help file discusses the PilRC Designer visual resource editor including a small tutorial, usage information, and a resource reference. This file is located in the CW for Palm OS Tools\PilRC Designer folder.

Palm Documentation

The documentation in the CW for Palm OS Support\Documentation\Palm OS 5.0 Docs folder is authored by Palm, Inc., and is the most concentrated set of information you can find for developing for the Palm OS. Below is a summary of each of the manuals you will find in your installation.

<i>Constructor for Palm OS</i>	This document discusses development with the Constructor resource-editing tool. Learn specifics about the various supported resources.
<i>Development Tools Guide</i>	Here you can find out about the tools that Palm manages and puts out as part of the SDK, including the Palm OS Emulator (POSE), Simulator, Palm Debugger, Palm Reporter, and the Overlay Tools.
<i>FEP Developer's Guide</i>	This document is intended for developers who wish to create a language front-end processor for a Palm OS device
<i>Palm OS 5 ARM Programming</i>	This manual is useful for developers who are interested in incorporating native ARM code within their 68K applications.
<i>Palm OS Companion (part 1)</i>	This pair of manuals is great overview material. A new Palm OS programmer would do well to read most of this first manual before ever writing a line of code. In part 1, learn about the Zen of Palm, how the event loop works, how memory and databases work, and get some strategies for debugging your applications.
<i>Palm OS Companion (part 2)</i>	This second companion manual discusses methods of data exchange, including beaming, serial communications, networks, internet connections, and telephony.
<i>Palm OS API Reference</i>	Once you begin your own development, this manual will be vital to your work. Here you find each API call documented fully, along with a Compatibility Guide that lets you know which version of the Palm OS is required for that API to be supported.
<i>Palm OS UI Guidelines</i>	This manual provides a description of how to design applications for Palm OS handhelds so that they conform to the Palm, Inc. user interface guidelines.

Sample Code

There is a significant amount of source code that is part of your CodeWarrior installation. This is a great place to see how to do something specific or to look at how a particular call is used. The (CodeWarrior Examples)\Palm OS\Palm OS 5.0 SDK Examples folder contains the applications provided by Palm as part of the SDK. This includes most of the built-in apps for you to use as a guide. The (CodeWarrior Examples)\Palm OS\Metrowerks Examples folder contains samples created or gathered by Metrowerks to supplement the examples from the SDK.

Knowledge Base

Palm maintains a very large searchable database of source code, FAQs, and articles written by a variety of people. You can search the KB at <http://kb.palmsource.com/> to find high-level and low-level discussions of how to properly code your application.

Newsgroups and Forums

Communicating with other Palm developers is a crucial part of developing for this platform. You may be new to development for this platform, but that doesn't mean you can't use the common pitfalls others have struggled through to make your life easier. The most commonly used newsgroups for Palm developers using CodeWarrior are the `codewarrior.palm` and `pilot.programmer.codewarrior` groups, both hosted through usenet. The highest volume of developer postings are found on the Palm Developer Forums described at <http://www.palmos.com/dev/tech/support/forums/>. These forums can be received as emails, a once-daily digest or viewed through a newsreader, and there's a forum for every part of the development cycle and for most of the Palm OS licensees. They are monitored by Palm engineers, Metrowerks engineers, and many experienced developers. Most beginning questions can be answered by a quick search through the archives, which are also referenced at this page.

Setting Up the Project

Let's get started with the project. In this chapter we will create the shell of our future application, and get an understanding of the development process with the CodeWarrior toolset. You will go through the following steps:

- Create the Project
- Modify the Stationery
- Compile, Debug, and Run
- Take a Look at the Code

Create the Project

Open the IDE to begin the process. Launch it from the Start menu or from the bin folder wherever you installed the tools. Then you need a *project file*. A project file is a file that helps you organize the development process. All your source files, resource files, libraries, and settings are stored in this project file. From this central point, you can edit your code and resources, compile and link the code, and debug the project. We will do all of these basic pieces in this chapter.

Now follow these steps to create a project file with the wizard.

1. Choose File-New from the IDE menu.
2. In the Project tab, select Palm OS C Application Wizard.

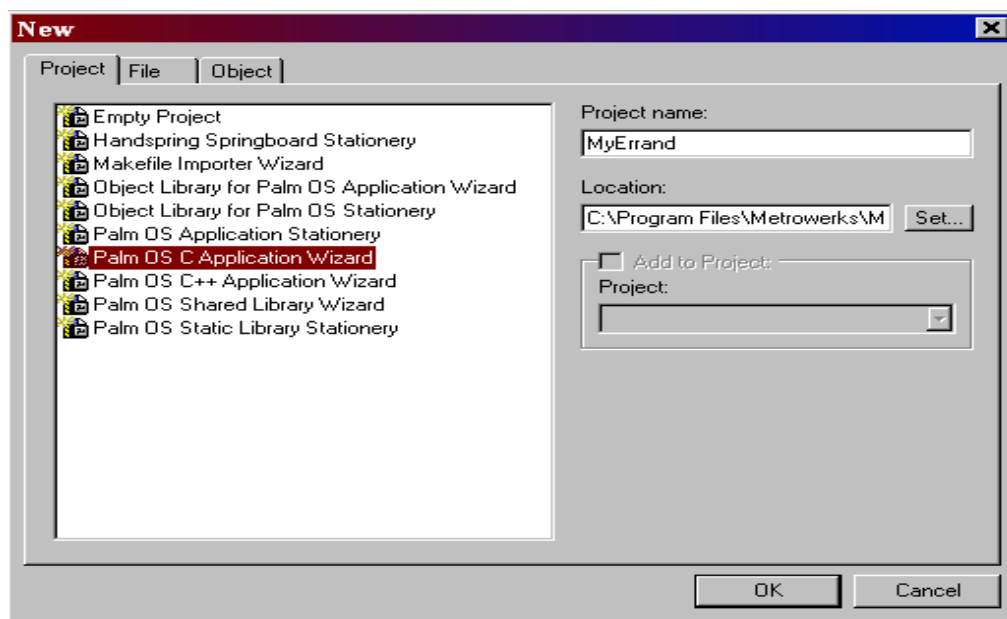


Figure 2.1: The New Project window

3. In the Project Name field, type the name of the project (this will also act as the name of the folder the project and all associated files will be stored in). Use MyErrand as the project name.
4. Set the Location of the MyErrand project folder to this Tutorial folder. If you used the default location when you installed CodeWarrior, this location should be C:\Program Files\Metrowerks\CodeWarrior\CodeWarrior Manuals\MW Tutorial For Palm OS\MyErrand. The New Project window should look like Figure 2.1
5. Click the OK button.

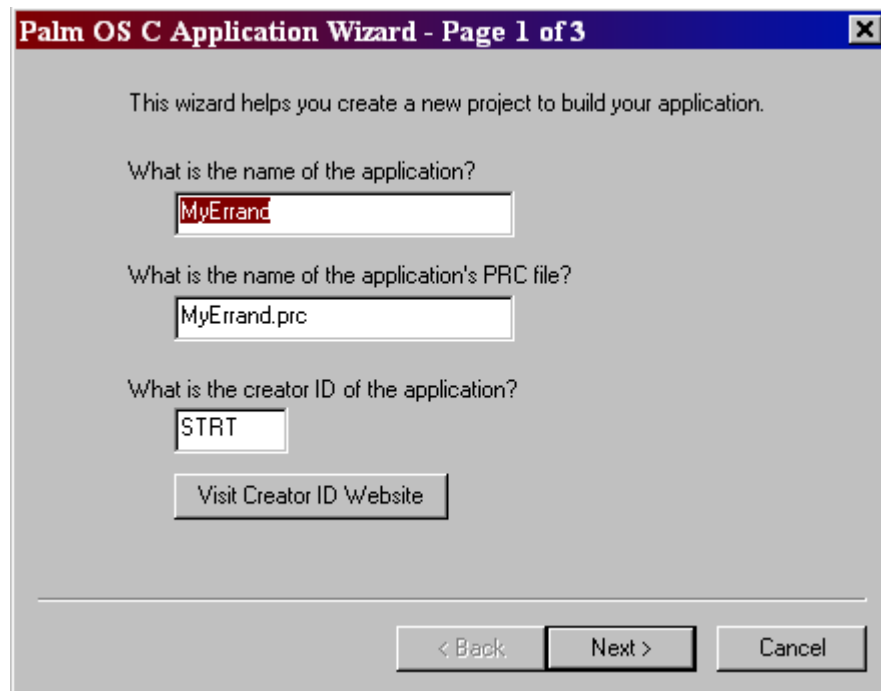


Figure 2.2: Page 1 of 3 of the Palm OS C Application Wizard

6. Leave the default values in the fields as they are shown in Figure 2.2 and select the Next button. Note that there is a Visit Creator ID Website button for registering new creator ids with Palm, Inc. Every Palm OS application, shared library, or feature has a unique four character Creator ID that distinguishes it to the Palm OS. However, you will only need to worry about obtaining a unique ID if you are planning to commercially distribute the application. You can ignore this for our purposes and leave the default value of 'STRT' in this field.

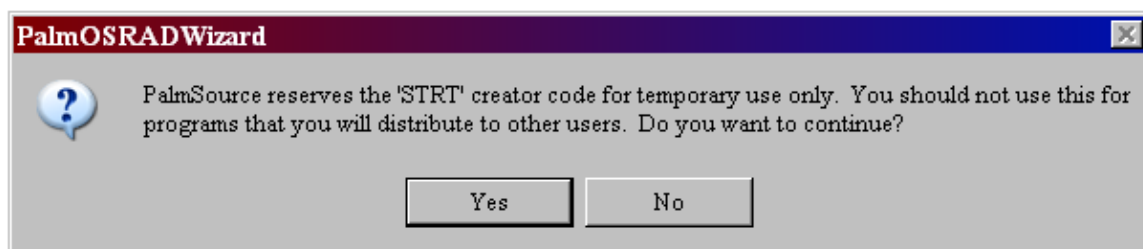


Figure 2.3: Palm 'STRT' Creator Code Alert

7. Click the Yes button in the Creator Code Alert dialog (Figure 2.3) that appears.

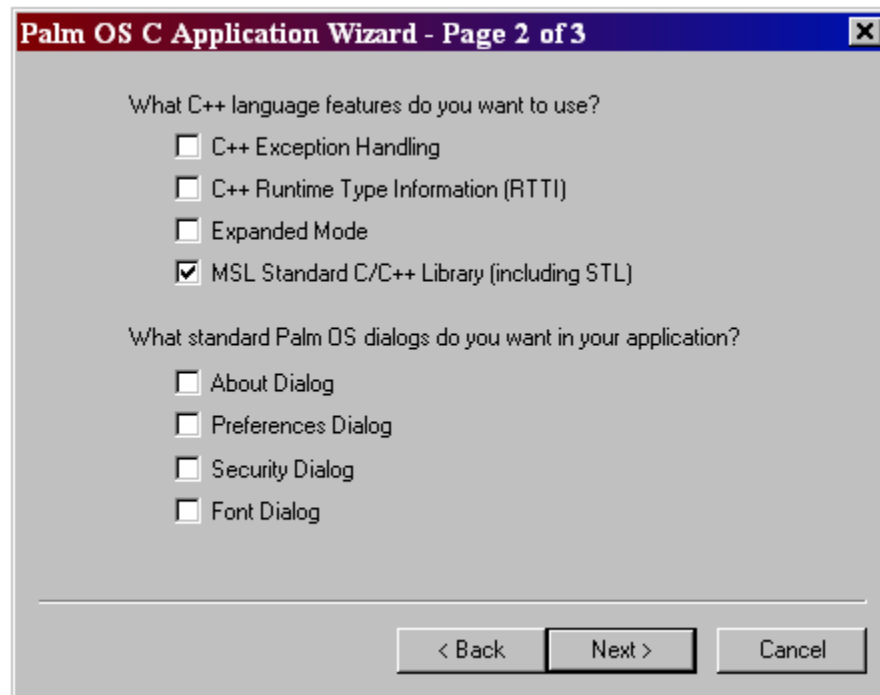


Figure 2.4: Page 2 of 3 of the Palm OS C Application Wizard

8. Leave the default values and click the Next button of the Page 2 dialog shown in Figure 2.4. We will not need to use any of the standard dialogs or C++ language features. However, if you would like to see how these Palm OS dialogs appear in the application, feel free to check the boxes next to the dialogs you wish to include.
9. The final dialog page of the wizard, which is shown in Figure 2.5, allows you to include support for third-party SDKs within your project. The following SDKs are available to choose from: Handspring, Symbol, Sony Clie, AlphaSmart, MathLib Floating Point Library, FontBucket from HandsHigh Software, and PrintBoy from Bachman software. Information regarding updates and where to find more developer information can be found in the Other SDKs.txt file that is located in the CW for Palm OS Support\ (Other SDKs) folder. Since we will not be needing any of these third-party SDKs, select the Finish button in order to generate the project for the application.

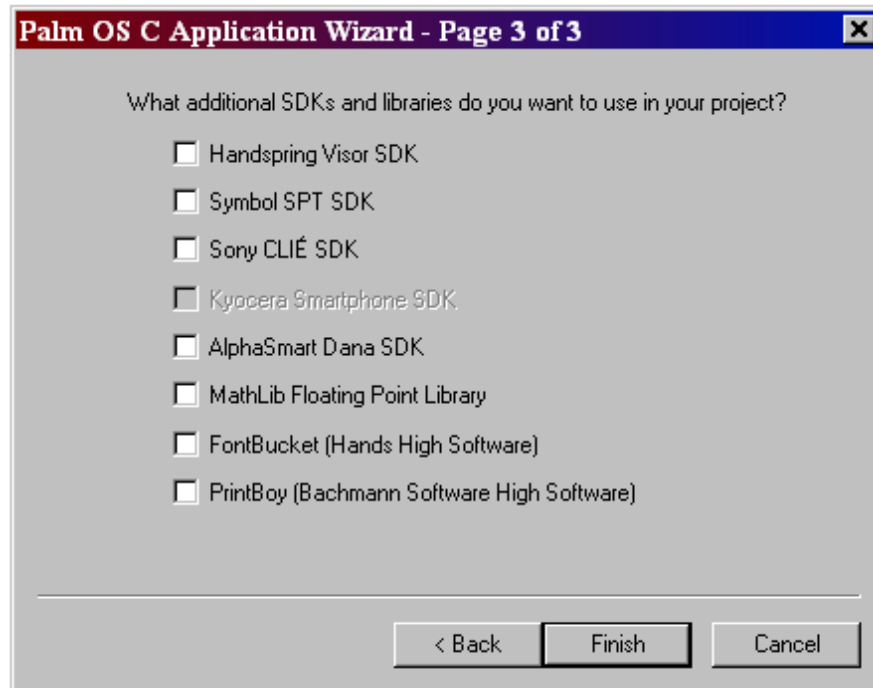


Figure 2.5: Page 3 of 3 of the Palm OS C Application Wizard

The project window can be viewed in three different formats within the IDE: docked, floating, and MDI Child. The dockable windows feature is available in Multiple Document Interface (MDI) mode only. This feature is not available in Floating Document Interface (FDI) mode. Toggle the Use Multiple Document Interface option in the IDE Extras preference panel to change between these two modes.

The project will be generated in docked view if you are using MDI. Once the project is generated, you can change views by right-clicking on the tab inside the docked window that represents the window you want to undock. More information on docking and undocking windows can be found in the [IDE 5.0 User Guide](#).

Try right-clicking and changing the view to MDI Child. The project window should look like Figure 2.6. If you expand the various folders (called groups from now on) in the Files tab, you will see several files. In the Source group, there is a MyErrand.c file, which is the source for this basic beginning to our project. The Headers group contains MyErrand.h, a header file for the project, and MyErrand_Rsc.h, the header generated for use with the resource file. In the Resources group, you see the MyErrand_Rsc.rcp file, which is the PilRC resource file. In the Libraries group, you see a Debug group and a Release group that contain the debug and release versions of the MSL C++ library for Palm OS, respectively. The Libraries group also contains PalmOSRuntime_2i_A5.lib. The last file you will see is readme.txt. This text file contains helpful information about the project such as the version of the Palm OS SDK that it will be using, which project files have been created, a list of the bitmaps that will be used for the application icon and whether certain application settings have been enabled or disabled.

Other features of this project window can be cryptic if you don't know what they are, and very helpful if you do. Here we will talk about some of the most commonly used

ones in Palm development, but you should refer to the [IDE 5.0 User Guide](#) for more information on some of the more powerful features of the project, targets, and the rest of the IDE. In Figure 2.6, you can see the project window as we discuss some of the features from there.

Tip: Here is a helpful tip for viewing files within the project window: hold the Ctrl key while clicking on the plus sign next to a group in order to open all groups within the project window; or hold Ctrl+Alt in order to open all groups and subgroups within the project window.

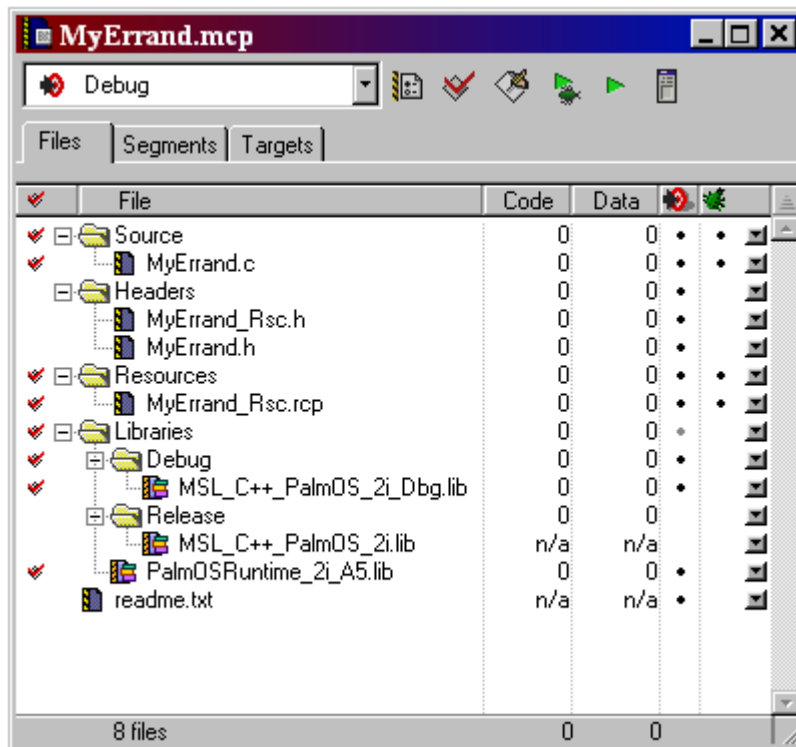


Figure 2.6: The Project Window in MDI Child View (all groups/subgroups open)



Red checkmarks next to the file names indicate that the file needs to be compiled.

The target dropdown shows you the current target and allows you to switch between targets in the project.

The settings button will take you to all the settings that are specific to the current target.

The make button compiles and links the project.

The debug button prompts the IDE to debug the application to the emulator or device.

The run button prompts the IDE to load the application on the emulator or device and launch it.

If you select the Segments tab, you can see how the files are broken down by segment. Segments allow you to create Palm applications larger than 64K. Each segment in a Palm OS application can contain up to nearly 64K of code. We will only have a single segment application in this tutorial, since this application is small. Another use of the Segments tab is to specify the link order. The files are linked from top to bottom in this list, and you can drag files around in this view to change that order.

Modify the Stationery

You've seen how to create the project and understand what it means, so we need to modify it for use in this tutorial. The settings we put in the wizard when we created the project are sufficient for this project. We do need to change the source and resource files that will be used, however.

First, we need to remove the files that are currently in the project, so we can replace them with our own.

1. Choose the Files tab on the project window.
2. Select MyErrand.c in the Source group.
3. Hit the "Delete" key or right-click on the filename and choose "Delete" from the menu.
4. Choose "OK" on the confirmation window to delete it from the project.
5. Repeat steps 1 through 3 with the MyErrand.h, MyErrand_Rsc.h, and MyErrand_Rsc.rcp files.

Now there isn't anything in the project to create an application. However, we haven't made any changes to what actually exists on the hard drive, so we could always add those files back into the project and have the wizard-generated app again. Instead, we want to get rid of those files, since they are placeholders we don't want to use. Open Windows Explorer and navigate to your CodeWarrior Manuals\MW Tutorial for Palm OS\MyErrand folder. Delete the contents of the Src and the Rsc folders. Leave the Src and Rsc folders on your disk so we can use this hierarchy throughout the tutorial.

In the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 2\Src folder, copy the Errand.c and Errand.h files and paste them in your CodeWarrior Manuals\MW Tutorial for Palm OS\MyErrand\Src folder. To tell the project about this new source file, either drag the Errand.c file from Explorer into the project window, or choose Project-Add Files and navigate to the CodeWarrior Manuals\MW Tutorial for Palm OS\MyErrand\Src folder and select Errand.c. On the Add Files dialog, make sure both the Release and Debug targets are selected and press OK. It is not necessary to add header files to the project as they are included by the source file, but add the Errand.h file anyway, because it will make it easier to view the contents of that file later.

For our own organization in this project, it is helpful to put the source and header files in appropriate groups. Click the Errand.c file once to select it, and drag it into the Source group as shown in Figure 2.4.

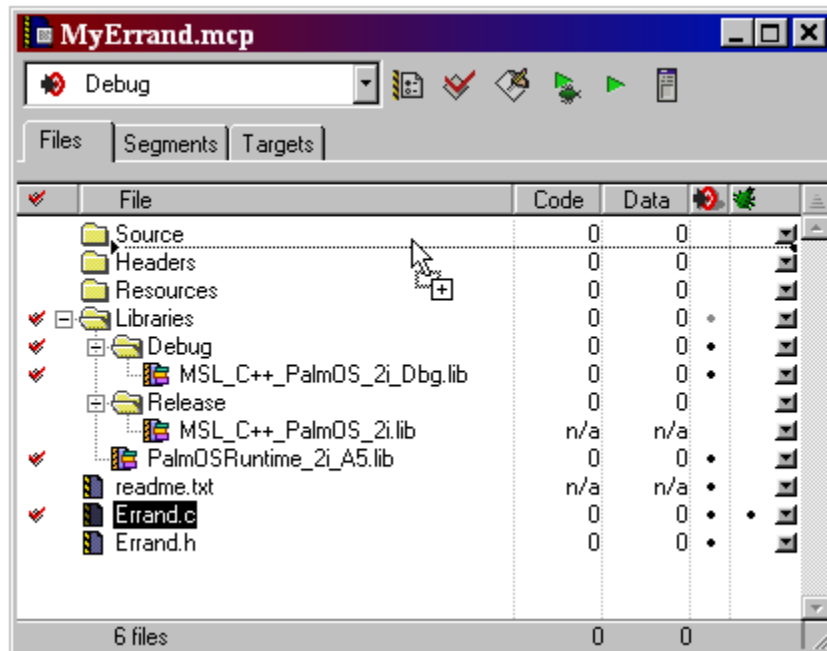


Figure 2.7: Moving a file into a group

Drag Errand.h into the Headers group. The Resources group is still empty, but we'll leave that be until Chapter 3. Meanwhile, this completes our modification of the stock stationery project.

Compile, Debug, and Run

Now let's compile and debug our project to make sure the environment is set up properly. First, compile and link the code to create the working application. Use the Make icon in the project window or choose Project-Make to compile the project. You should see a progress window appear giving you information about the compile and link process. The project should compile and link without errors or warnings. If not, go back to the beginning of this chapter and go through the steps again, to make sure you haven't missed anything.

To debug the application, you will need either a device running the Palm OS or a ROM file, which you can get from a device or off websites of companies that distribute devices with the Palm OS. There are big advantages to using ROM files instead of actual devices – you can test your application on many different devices without having to spend a gazillion dollars, and you get improved error checking through the Palm OS Emulator (a free tool available from Palm, called POSE from here on out) and debug versions of the ROMs on the websites. To get these ROM files, see below for a list of vendors and the appropriate websites:

Palm	http://www.palmos.com/dev/tools/emulator/#roms
Handspring	http://www.handspring.com/developers/tech_pose.jhtml
Sony	http://www.us.sonypdadev.com/top.html
Handera	http://www.handera.com/support/developsupport.asp
Symbol	http://software.symbol.com/devzone/
Kyocera	http://www.kyocera-wireless.com/partner/partner.htm

In order to get access to the ROM files, you will need to sign an agreement with any/all of these vendors. Even if you have a device to debug to now, you will need to get these ROM files to test your future applications to any reasonable degree.

In this tutorial, we will assume you are debugging to POSE with a ROM file you uploaded from a device or downloaded from one of these vendors' websites. If you are debugging directly to a device, refer to the [Targeting Palm OS Platform](#) manual for step-by-step directions for making that connection.

If you would like to use the Palm OS 5 Simulator, please refer to the Palm OS 5 Simulator documentation, which is located in the following folder of your CodeWarrior Developer Studio for Palm OS, V9 installation directory (debug version): CW for Palm OS Tools\Palm OS 5 Simulator\Palm_OS_5_Simulator_Dbg_dr12\Documentation.

To start up POSE, we need to need to configure a couple of things. In the IDE under the Edit menu, choose Debug Settings. Select the Palm OS Debugging Settings panel, which is the last panel under "Debugger" in the tree list.

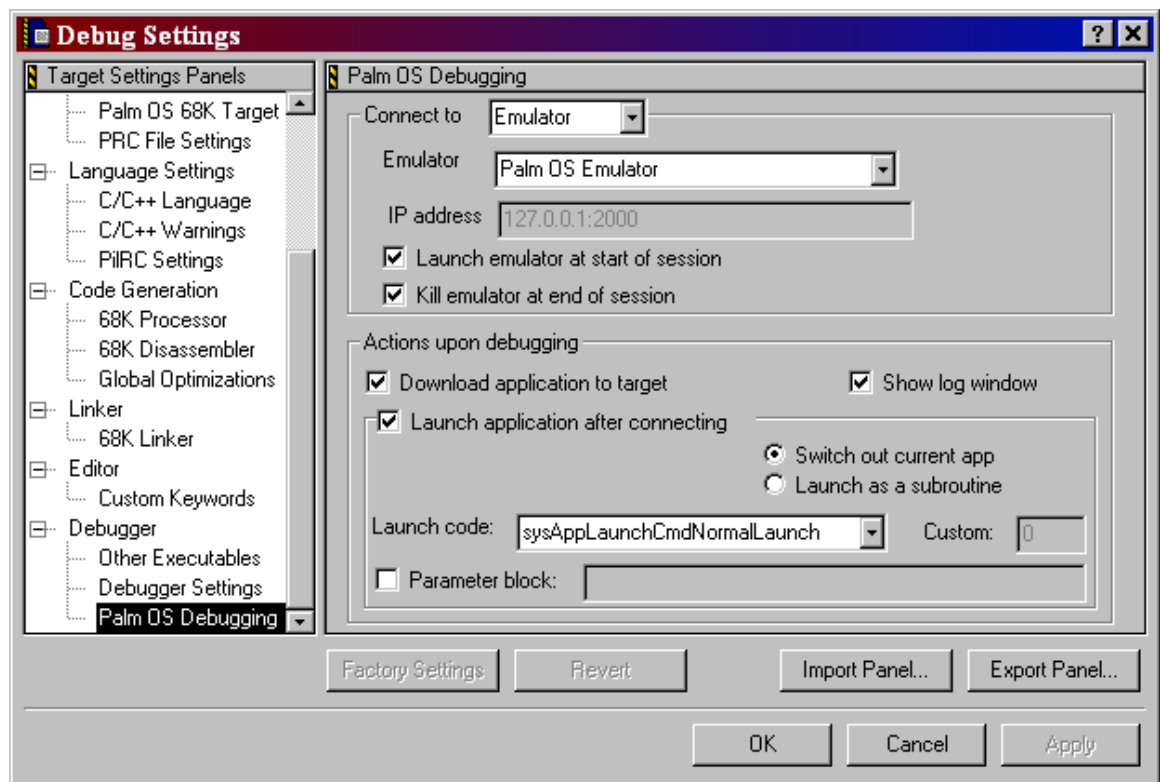


Figure 2.8: The Palm OS Debugging Settings panel configured for POSE debugging

If you check the box next to Launch emulator at start of session, then when you start the debug session, the IDE will find the specified POSE at the default location within the CodeWarrior for Palm OS installation directory (CW for Palm OS Tools). Although Palm OS Emulator is the default emulator, you can select a different type of POSE depending the type of ROM you wish to use. For instance, if you have a Sony Clie ROM, you can use the Clie POSE by selecting Palm OS Emulator (Sony Clie) from the Emulator pull down list. In this tutorial, we assume that you will be using the default POSE. When you're done, click the Apply button, and the panel should look like Figure 2.8.

*Please refer to the [Targeting Palm OS Platform](#) manual for more information on this panel.

Click OK to close the settings panel and the target settings window. Since we have not started a debug session yet, POSE is not running. Another way to start it is to choose PalmOS->Launch Emulator from the IDE menu bar. If you've never run POSE before, you will see the dialog in Figure 2.6.

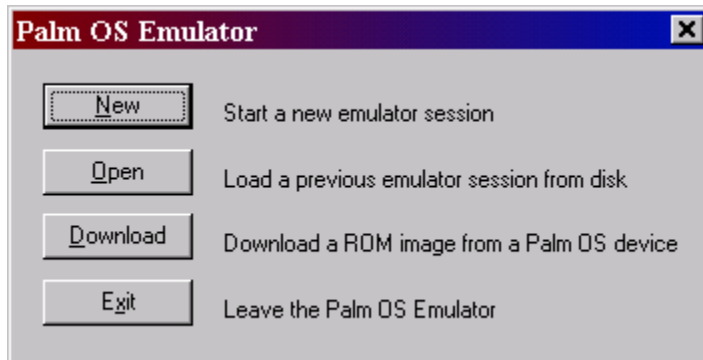


Figure 2.9: The startup POSE dialog

Select New to get the New Session dialog in Figure 2.10. The ROM File field will say None, and you can select it a choose Other to navigate to a ROM file you have stored on your machine. The Device field will then limit you to devices the chosen ROM file can run on. For the Skin, use Standard-English

or Standard-Japanese to see pretty pictures of the selected devices (if you choose Generic for the Skin, you will always see a PalmPilot Pro-like picture, but it won't affect the way your program interacts with the chosen device). For the RAM size, select something that makes sense for the ROM file you have chosen. Most devices have 2 or 8 Megs of RAM.

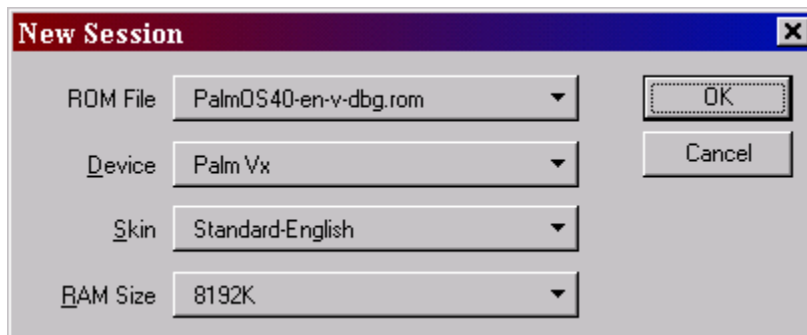


Figure 2.10: The POSE New Session dialog

Once you click OK, POSE should come up, looking like the device you selected. When POSE is done launching, it should be displaying the Preferences screen, which is an ideal state for the device to be in when debugging to it.

To debug the application, simply press the debug icon on the project window or choose Project-Debug from the IDE menu. This will bring up the debugger, stopped at an automatic breakpoint at `PilotMain()`. You can compare what you see to what you should see by looking at Figure 2.11.

You should also see a debugger log window appear that shows useful information about the debug session such as the IP address and other debug process data.

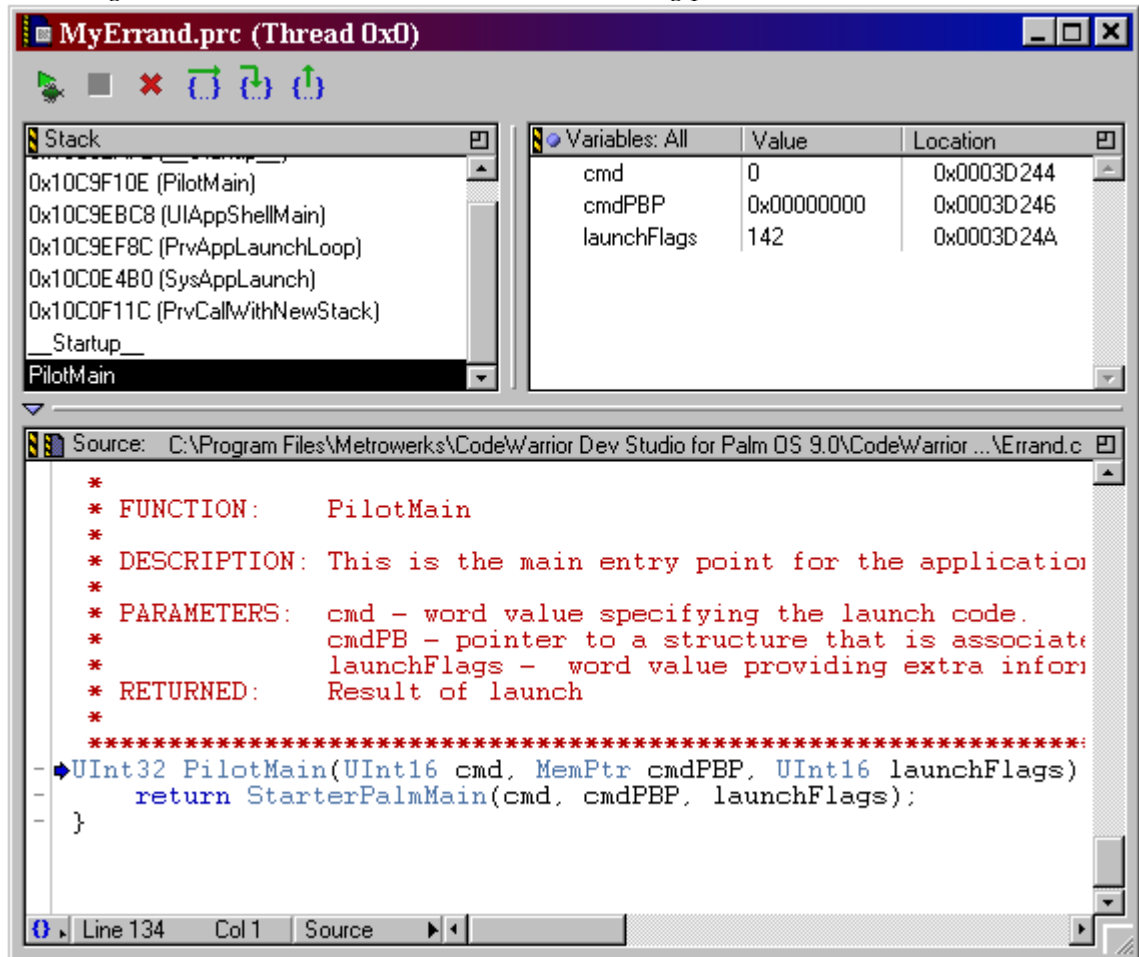



Figure 2.11: The debugger window at the beginning of a debug session

In order to familiarize yourself with the debugger window and the commands that are available, you should refer to the debugger section of the [Targeting Palm OS Platform](#) manual and the [IDE 5.0 User Guide](#). To debug the application to the next breakpoint (we haven't set any, so it will run the application normally), click the Debug icon in the debugger window. The debugger window will go blank, and POSE should look blank, too. Kill the debugger by clicking the red X next to the debug icon. This will also reset POSE.

Once POSE is finished resetting, let's run the application two ways. First, click the App

Launcher icon  in POSE to view the applications on the device. You should notice one called MyErrand. Click it with your mouse to launch it. You will see the same blank screen you saw when you debugged the application. Click the App Launcher icon again to get out of the app. Another way to run the MyErrand application is from the project window. Back in the IDE, click the Run icon, and you should see the same blank screen you've seen twice before.

You may have realized that this application doesn't do anything yet. We need to add functionality to it in future chapters.

Take a Look at the Code

Let's look at the code in this very simplistic example. As you've already seen, this application doesn't actually do anything, but looking at the code will give you an idea of the flow of a Palm application. If you double-click on the Errand.c file, it will open in the CodeWarrior editor for you to look at. Using the function pop-up menu in the source window depicted in Figure 2.12, jump to the PilotMain function, and that's where we will begin.

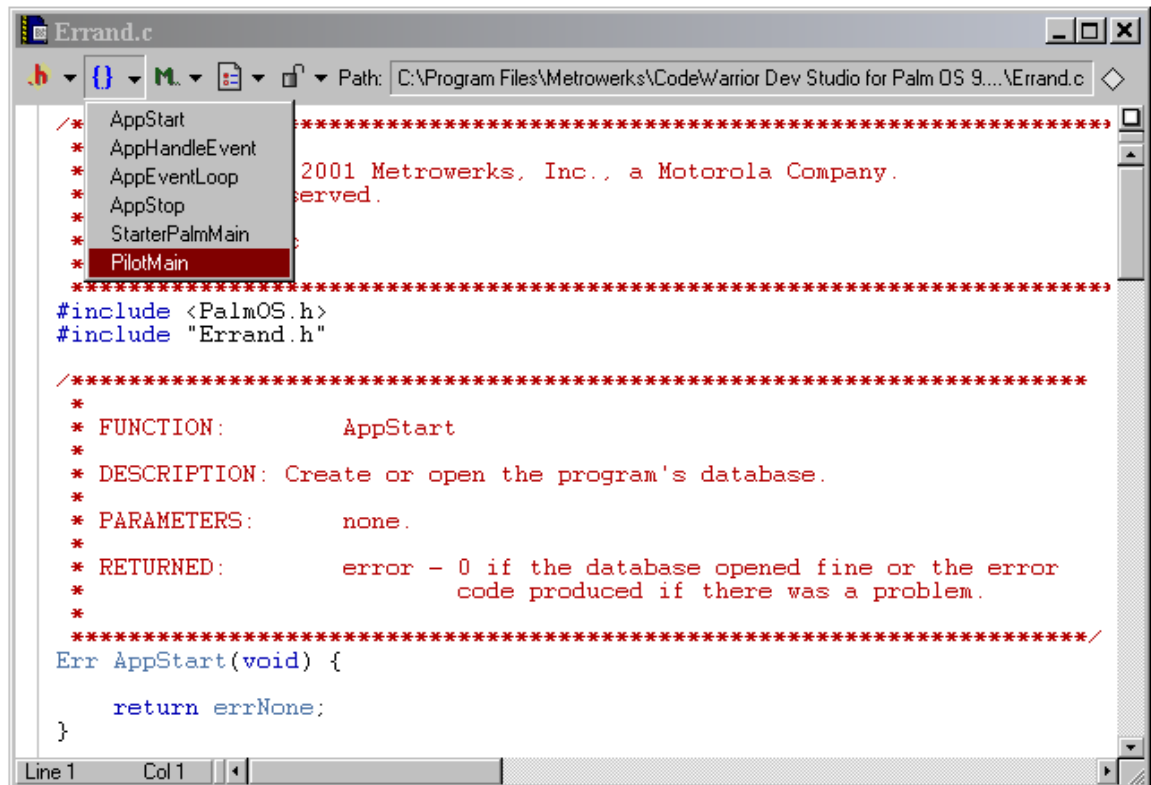


Figure 2.12: Using the function pop-up menu

You will notice that the function names are colored in a different color than the standard black. This is because of the browser (see the [IDE User Guide](#) for more information on this feature). Besides turning the text pretty colors, this will make it easier to navigate the code. If you right-click on a function name, you will see an option to go to the function definition of that file.

If you look at the code, you will see that PilotMain calls StarterPalmMain. If you right-click on StarterPalmMain and jump to the function definition as shown in Figure 2.13, you will be able to easily navigate through the code as we discuss it.

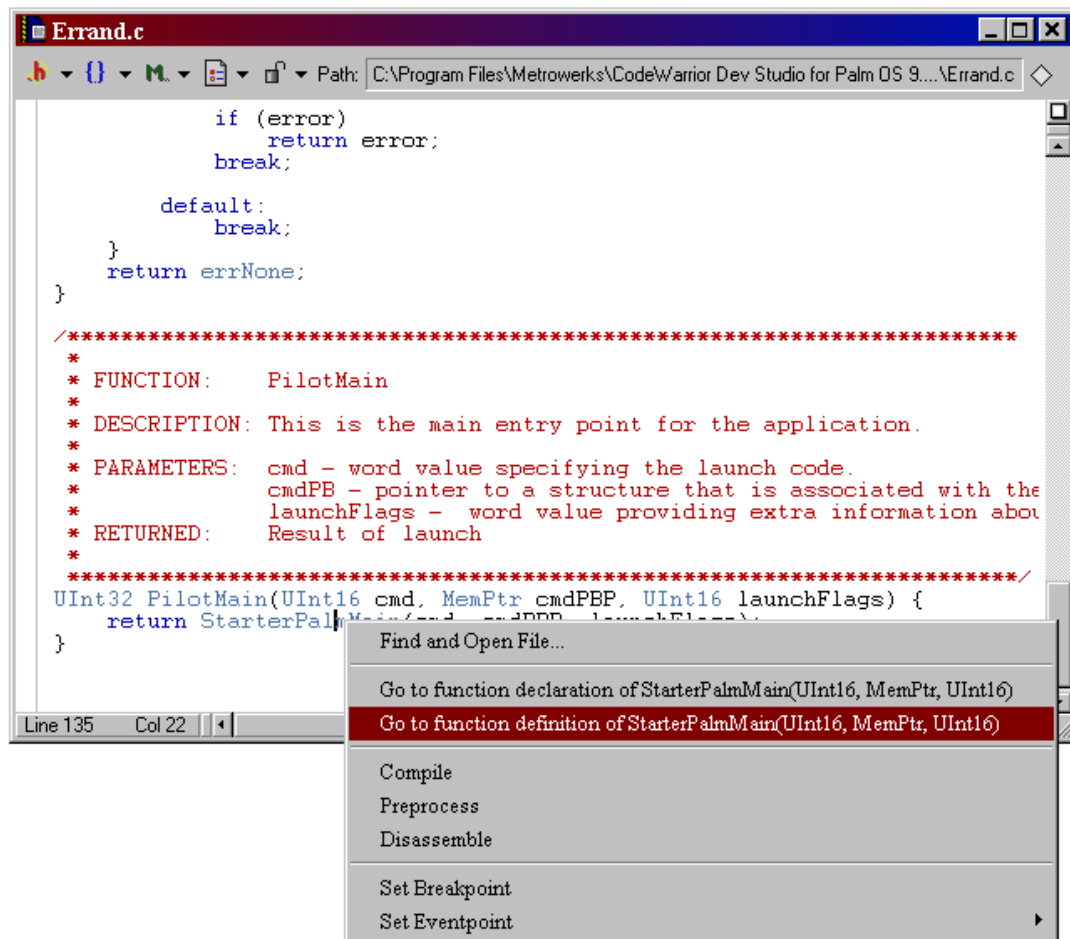


Figure 2.13: Navigating using browser coloring

When MyErrand is launched, a launch code is sent to PilotMain. Our PilotMain function then sends that launch code to StarterPalmMain. In the StarterPalmMain function, we see that the only launch command we do anything with is the `sysAppLaunchCmdNormalLaunch` command – the one the app gets if a user specifically launches the application. From the case for this launch command, `AppStart` gets called. If `AppStart` has no error (which it won't in this little sample), then the `AppEventLoop` is called from `StarterPalmMain`. The `AppEventLoop` is the control center of a Palm OS application. It gets new events from the user and asks the system, then the menu handler, then the application, and then the form handler to handle the event that came in. The application gets the chance to handle the event through the `AppHandleEvent` function. Right now our application doesn't do anything, so `AppHandleEvent` always tells the OS it didn't handle any events and returns false. This will change as we add things in future chapters. Once an `appStopEvent` is queued up (like when the user taps an icon or presses a hard key to go to another application), we exit the `AppEventLoop`, and go back to `StarterPalmMain`. Then `AppStop` is called, and we clean up any mess our application has made. The application returns from `AppStop` to `StarterPalmMain` to `PilotMain`, and then it is finished.

Most Palm OS applications are significantly more complicated than this, but most follow this same set of basic steps.

A Form and an Alert

Now that we have the base project created, we're ready to get down to the creation of some resources. First, get the source files you'll need for the new resources we create in Chapter 3 to work properly.

- ❖ If you followed Chapter 2, go into the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 3\Src folder and copy the Errand.c, Errand.h, and Main.c files to your CodeWarrior Manuals\MW Tutorial for Palm OS\MyErrand\Src folder, overwriting the source files already there.
- ❖ If you skipped Chapter 2, go to your CodeWarrior Manuals\MW Tutorial for Palm OS folder and create a folder called MyErrand. Copy all the files and folders in the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 3 folder to your MyErrand folder.

Open the project, and add the Main.c file to the project. Put this file in the Source group.

In this chapter, you will go through the following steps:

- Set up the Resource File
- Create a Form
- Create an Alert
- Compile and Run
- Take a Look at the Code

Set up the Resource File

To create a form, we need a resource file, and we're going to use PilRC Designer to make that file. You can open PilRC Designer directly from the IDE by choosing PalmOS->Launch PilRC Designer from the IDE menu bar. In the File menu, choose New or hit Ctrl+N. This will create a new empty document in the editor and the Design View layout will open so you can start editing resources in the Visual Designer. There are two main views for each resource document, a Visual Designer, for graphically editing controls and resource elements and a Text Editor, for editing the actual textual source code for the resources.

There are three main windows that you will see in the Visual Designer: the Property Explorer (the leftmost window), the visual Design View layout (the center window), and the Control Palette (the rightmost window). You will discover the purpose and use of these windows as we continue through this chapter.

First, we will save the resource file and the corresponding resource header file. The resource header file will contain the constant resource IDs for each resource that is contained within the resource file. Switch to the Source View by selecting the Source tab in the bottom portion of the editor. This will take you to the Text Editor, which lets you edit the resource script directly outside the Visual Designer. Changes made in the Text

Editor will be reflected in the Visual Designer when switching back to the Visual Designer. When creating a new document, no header file will be included, so it's up to you to insert one. In the Source View, insert the following line at the beginning of the document:

```
#include "ErrandRsc.h"
```

Save the resource file by selecting File->Save As. Enter the file name as Errand.rcp and make sure to save the file in your MyErrand\Rsc folder in order to maintain the organizational structure. After the resource file has been saved, the Designer will ask you if you want the header file to be created. Select the Yes button and the Designer will create a new header file in the same directory the resource file. The header file will not get created unless you save the resource file first. This process would have also worked if you had saved the resource file first, entered the #include statement in the Source View, and then switched back to the Design View.

We will be using the Design View throughout the remainder of this tutorial for creating and manipulating resource elements. However, since these are text based resource files, feel free to look at the Source View from time to time if you are interested in seeing how the graphical resources we create are represented in text format.

Now you are ready to go back to the IDE. Use Project->Add files to add Errand.rcp and ErrandRsc.h to the project. Be sure to add the files to both targets. For our organizational structure, drag Errand.rcp to the Resources group and ErrandRsc.h to the Headers group. After all this, your project window should look like Figure 3.1.

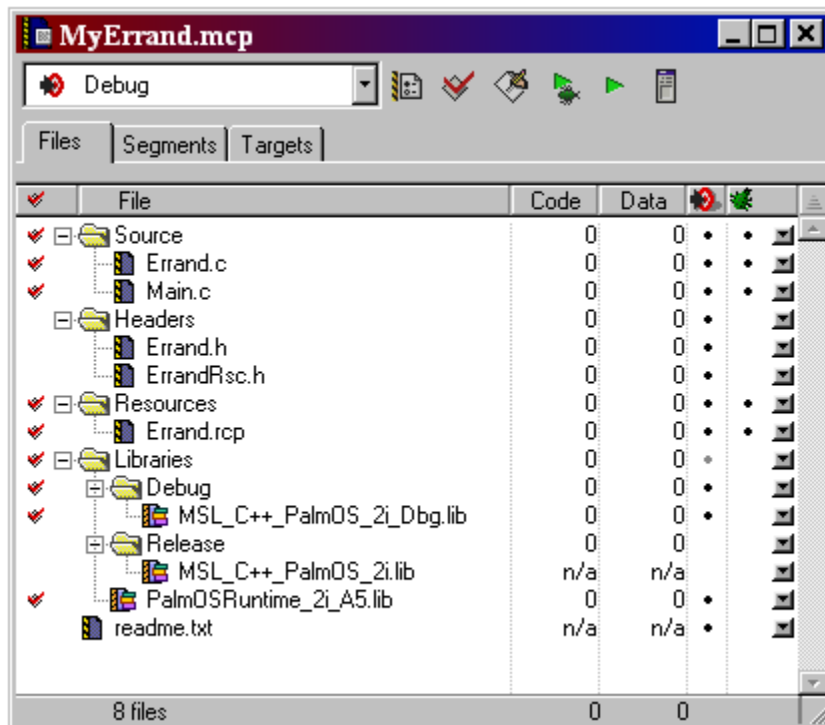


Figure 3.1: Project window with the resource files added

Tip: PilRC Designer supports the GENERATEHEADER command that will automatically generate the header file when the resource file gets compiled by the PilRC compiler. There is a GENERATEHEADER command under the System category of the Control Palette that adds the command to the Source View of your resource file. After clicking on GENERATEHEADER, all you have to do is change to the Source View and rename the header file from the default “headerfile.h”.

Create a Form

In PilRC Designer, go back to the Design View for the Errand.rcp file that we just created. We are now ready to create the first form.

1. From the Control Palette, select FORM from the Form category of resource items. This will create a new form resource and you should see a preview of the form in the Design View layout.
2. For a preview of how the form will look on a device, use the skin selector that is located in the top portion of the Design View. These are the same skins that are used by the Palm OS Emulator. Although the Emulator is needed for the Designer, the skins are used to display images of various devices. The PilRC Designer automatically parses the CW for Palm OS Tools\PilRC Designer\Skins folder and its subfolders.

Figure 3.2 shows the form display and the Control Palette.

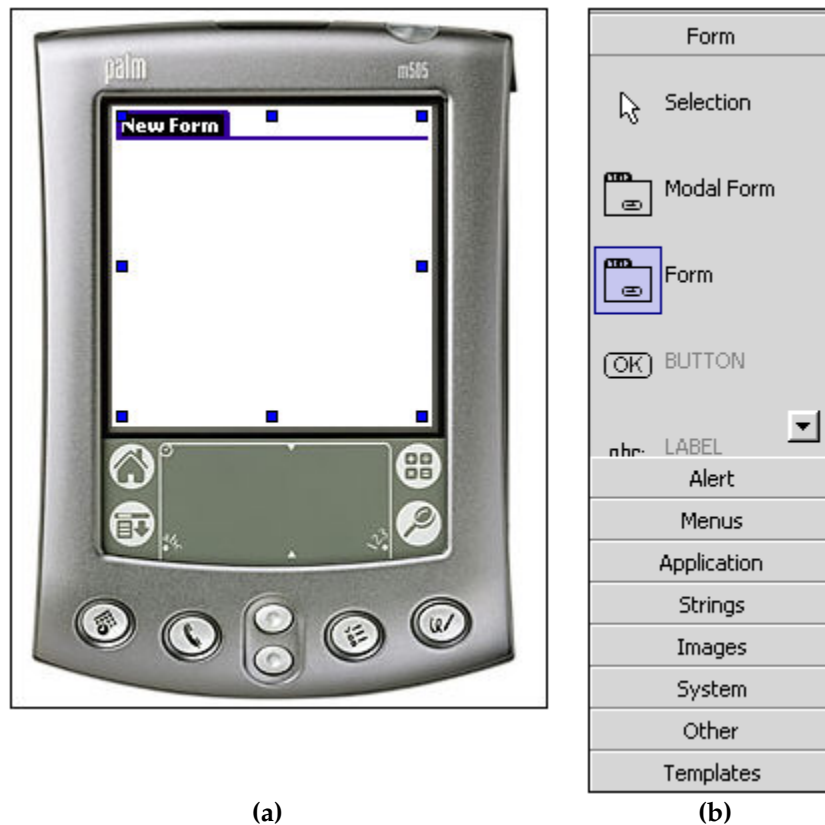




Figure 3.2: (a) The new form with the M505 skin selected. (b) The Control Palette showing the Form controls.

3. Select the form control by clicking on the form title, "New Form", in the Design View layout display. You can zoom in or out in order to increase or decrease the viewing size of the Design View layout by selecting  (Zoom In) or  (Zoom Out) from the Designer's toolbar.
4. In the Property Explorer window, shown in Figure 3.3, select the ID field and replace "AUTOID" with "MainForm". The Property Explorer, or Property Editor, displays the editable property settings of the selected control. From this point onward, the term Property Explorer will be used to describe the entire window, and Property Editor will be used to describe the field properties that get displayed inside this window for different resource types.
5. Select the Title property field and change the value from "New Form" to "Errand Manager". All changes will be updated immediately in the layout after they are made.
6. Make sure the Height and Width fields are set to 160 each and the Left and Top fields are set to 0.

Tip: The Control Palette uses a set of template files that define each of the resource types that it lists. It is possible to define your own templates to add special forms or resources you use often to the Control Palette by either adding controls to an existing file, or by creating your new file. The template files end in a '.ctl' extension and are located in the Library/PalmControls folder of the PilRC Designer directory. The file format is XML and it is documented in the [PilRC Designer Help](#).

Misc	
ID	MainForm
Savebehind	<input type="checkbox"/>
Usable	<input checked="" type="checkbox"/>
Other	
DefaultButton	
Frame	pfNoframe
HelpString	
Menu	
Modal	<input type="checkbox"/>
Title	Errand Manager
Position	
Height	160
Left	0
Top	0
Width	160

The following Property Editors are available for different kinds of properties:

1. Text
2. Dropdown list
3. File open dialog
4. Font Editor
5. Item Editor

Each Property Editor will automatically be displayed for the correct property type.


The other property fields can be left as they are. The Property Editor should look like Figure 3.3 after you have made the changes.

Figure 3.3: The Property Editor after updating the ID and the Title.

We have finished creating the form and are now ready to create the alert.

Create an Alert

Alerts are similar to forms in several ways. However, they are enough different to warrant their own resource type. Now that we have our form, let's create the alert.

1. Select the Alert category from the Control Palette in the Designer.
2. Select the ALERT control from the Alert category. This will create a new alert resource and it will appear in the design preview.
3. Select the ID field in the Property Editor for the Alert and replace "AUTOID" with "RomIncompatibleAlert".
4. Select the AlertType field and change the value to "ppError" by selecting it from the pull-down menu.
5. Select the Buttons field. Delete the Cancel button. Open the String Editor window by clicking the open button () next to "(TStrings)" or by double-clicking on "(TStrings)". In the String Editor window that appears, delete "Cancel" from the display. Move the cursor to the same line as the last string in the Editor window ("OK") and hit the OK button to close the window. If you forget to position the cursor on the same line as that of the OK string, then a button with an empty string will get created. If this happens, reopen the String Editor window and hit the OK button once again or reposition the cursor and then hit OK.
6. Select the Message field and enter the following: "System Version 2.0 or greater is required to run this application."
7. Finally, select the Title field and enter "ROM Incompatible".
8. Leave the default values for the remaining properties and save the resource file.

Once you've completed this, your Alert Properties should look like Figure 3.4.

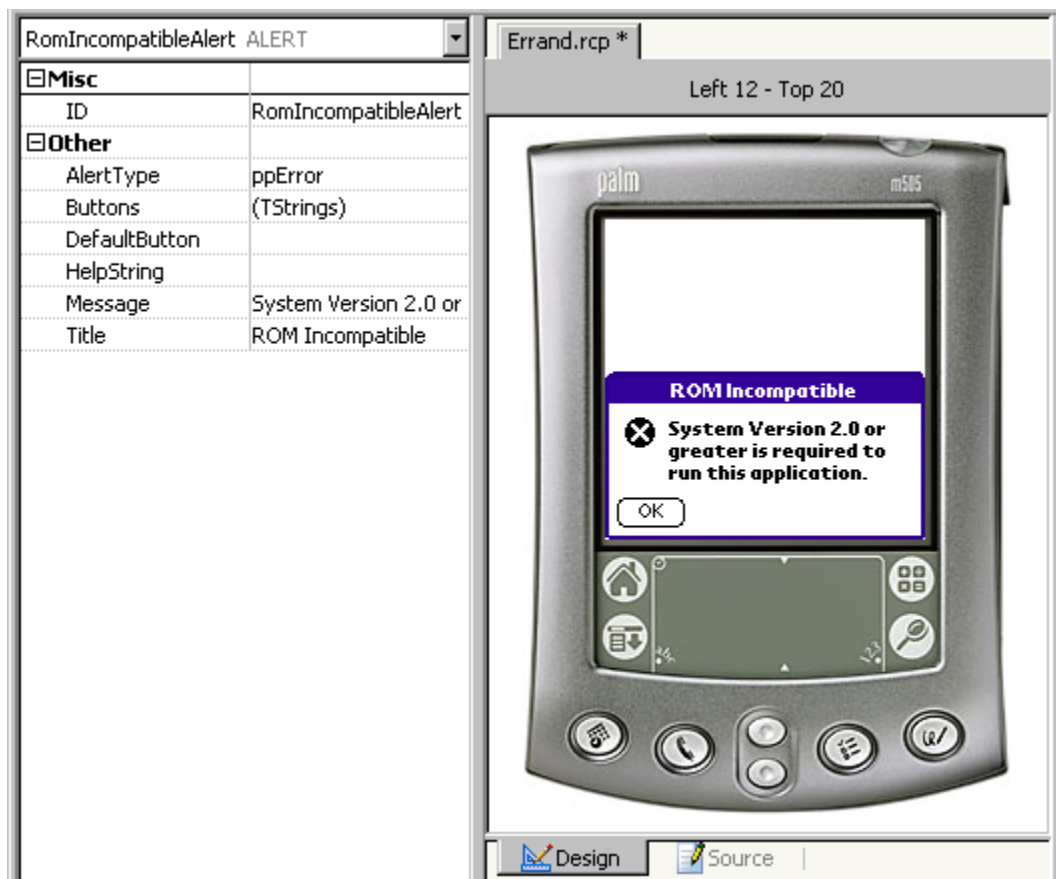


Figure 3.4: The ROM Incompatible alert in the Visual Designer

Switching Between Resources

In order to switch between the different resources within your resource file, use the Object Tree of the Designer (also referred to in this tutorial as the Objects list). The object tree shows a hierarchical view of all the resources that the file contains. In the Design View, the object that is selected in the tree will get displayed in the preview. In the Source View, you can jump to different object source locations by selecting the object in the tree.

From the Design View, open the Object Tree by selecting the Objects popup list located at the top portion of the Design View window. The object tree should contain the two resources we just created and should look like Figure 3.5.

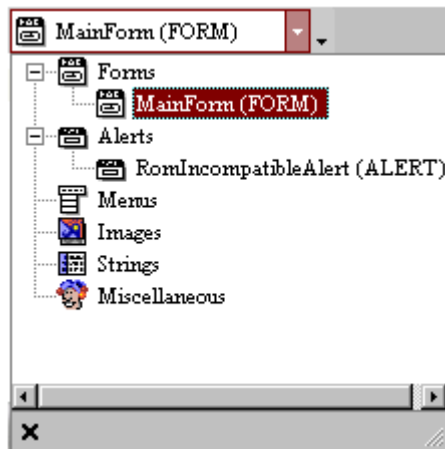


Figure 3.5: Object Tree after adding Form and Alert

We can now save the resource file and return to the IDE to test the new application.

Compile and Run

In the IDE, use the Make button or Project-Make to compile and link the project. If you get an error about an undefined identifier, go back into the resource file and make sure all of the resource values are typed correctly. Spacing doesn't matter, but case does. The code we copied over at the beginning of the chapter expects the resources to be named a certain way.

Before running the application, you should close the resource file in PilRC Designer or else, the Designer will detect that the file has been opened outside of the Designer and will try to reload the resource file when you return to the Designer, which can cause potential problems. Also, it is important that no application is running on POSE (including earlier versions of this application). In POSE, make sure the device screen is displaying the App Launcher (Figure 3.6) or the Preferences screen (Figure 3.7). You can


get to the App Launcher by pressing the App Launcher silkscreen button (). Get to the Preferences screen by running the Prefs application, or by resetting the device with any of the reset types discussed in Chapter 1.



Figure 3.6: App Launcher screen

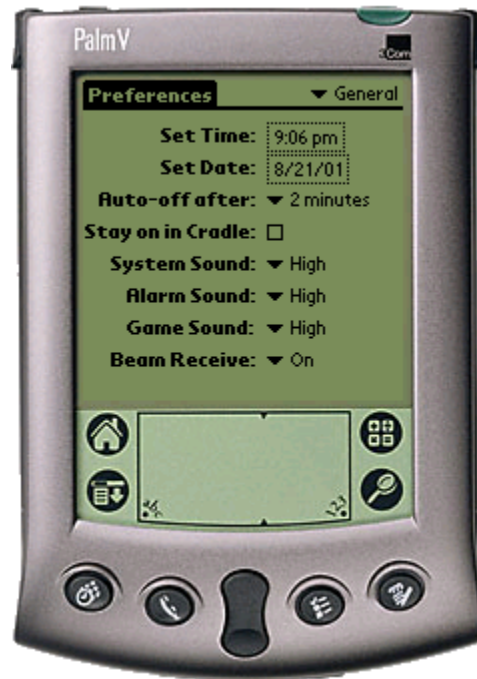


Figure 3.7: Preferences screen

Now, on the project window, choose the Run button to start the application on POSE. If you run the application on a device/ROM image that runs a version of the Palm OS before version 2.0, you will see the alert we created, as in Figure 3.8. Clicking OK on that alert will return you to the App Launcher. If you run the application on a newer version of the OS (something with 2.0 or later), you will see the form we created, as in Figure 3.9. Tap on the App Launcher to leave the application.

Right now our app doesn't do much, but you can see how it displays the resources we created in Constructor. As an additional exercise, you could debug the app (instead of just running it), and step through the code to see how the code interacts with the resources. This interaction is discussed in detail in the next section, where we "Take a Look at the Code".



Figure 3.8: App running on device with OS 1.0

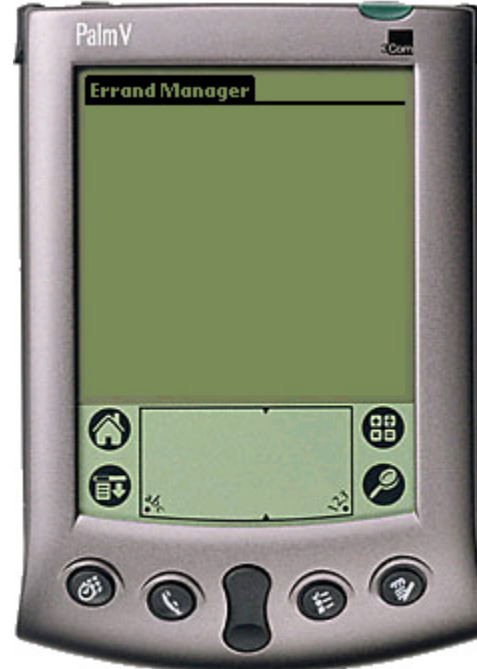


Figure 3.9: App running on device with OS greater than 1.0

Take a Look at the Code

There are several files that have been modified to make these resources work properly. We are going to look at those four files now to see the code that was created behind the scenes. Double-click on the filename in the IDE project window to open the file in the editor.

ErrandRsc.h

This is the header file that PilRC Designer creates for us. There are a couple of pieces that should be explained, so you understand why these values are here.

After the comment header, you see a `#define` that refers to the form resource with ID 1001. This was the default ID that assigned by PilRC Designer to the MainForm. You should see:

```
#define MainForm 1001,
```

which is the line that defines the constant MainForm to reference this resource that has ID of 1001. The “MainForm” identifier is then used throughout the source code to reference this resource. This name came from the ID field that we entered within the Property Editor within the Designer. Since the values we enter into the property fields of the Designer are used as constant resource identifiers within the source code, the values must be valid ANSI C strings, i.e. no spaces or other invalid characters. However, if you do enter an invalid character, PilRC Designer will automatically change the value into a valid string by replacing the invalid character with an underscore. For example, if you entered “Main Form”, it would have been changed to “Main_Form”.

The next `#define` grouping is about the alert resource with ID of 1000. The `RomIncompatibleAlert` is the constant for the alert itself. Close this file, and continue through the rest of the code discussion.

Errand.c

Open the Errand.c file, and take a look at some of the modifications that have been made. Starting in StarterPalmMain, notice that the code now calls an additional function, RomVersionCompatible during the sysAppLaunchCmdNormalLaunch case. If you navigate to this function, you will see that it finds the version of the OS the device is running, and if it's less than 2.0, calls FrmAlert to display the RomIncompatibleAlert we created. Then, due to a difference in the way the OS worked in version 1.0, the function calls AppLaunchWithCommand to switch to the App Launcher manually once the alert is dismissed.

The next difference in StarterPalmMain is that we call FrmGotoForm(MainForm) before going into the event loop. This tells the application which form to start with (our Main form). Now, when the AppEventLoop calls our AppHandleEvent function, there is one event that we can handle – the one to display our form.

In AppHandleEvent, we receive the frmLoadEvent for the form in question. Then we call FrmInitForm and FrmSetActiveForm to setup the form for display. Probably the most important part of this code, though, is the switch statement that sets the event handler for the individual form. We call FrmSetEventHandler to do this, and give it the name of the function that will deal with events on this particular form. We've defined the function MainFormHandleEvent, which we will discuss in the section on Main.c in a moment. In the AppEventLoop function, the system, menu, and application all get a chance to handle an event. But if none of these handlers handle the event, then FrmDispatchEvent is called. FrmDispatchEvent calls the current form event handler, and then the form is allowed to work with the event.

In AppStop, we have added a call to FrmCloseAllForms. This is necessary to release the memory referring to our open form(s), so we can exit the application cleanly. Close this file and move on to the next one in our discussion.

Main.c

One of the standard methods of breaking a Palm OS application into separate source files is to create a separate source file for each form. This is what we have done in creating the Main.c file. This file is set up to handle anything that has to do with the Main form that we created in Constructor. In Errand.c, you saw that we set the form handler for the Main form to MainFormHandleEvent, and that is the only function in this file at this time. This function is called anytime FrmDispatchEvent is called from the AppEventLoop function while MainForm is the current form. Within the MainFormHandleEvent function, the only event type we handle at this point is one to open the form (a frmOpenEvent). So we use FrmGetActiveForm to get a pointer to the form resource, and pass that to FrmDrawForm to actually display the form.

Close this file and go on to the next and last file that was modified.

Errand.h

In the header file for the project, you can see that we have added prototypes for the RomVersionCompatible function in Errand.c and the MainFormHandleEvent function in Main.c. Beyond that, all the other components are exactly as we had them in the previous chapter.

Two Buttons and Another Form

Now it's time to make the application do something. First, copy over the source files you'll need to handle the new resources we're about to create.

- ❖ If you followed Chapter 3, go into the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 4\Src folder and copy the Errand.c, Errand.h, Main.c, and Create.c files to your CodeWarrior Manuals\MW Tutorial for Palm OS\MyErrand\Src folder, overwriting the source files already there.
- ❖ If you skipped Chapter 3, make sure you have a MyErrand folder created in the CodeWarrior Manuals\MW Tutorial for Palm OS folder. Then go to the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 4 folder and copy all the files and folders from that folder into your MyErrand folder.

Open the project, and add the Create.c file to the project. Put this file in the Source group.

In this chapter, you will go through the following steps:

- Create a Button on the Main Form
- Create a New Form
- Put a Button on the Create Form
- Compile and Run
- Take a Look at the Code

To get started, open the Errand.rcp file in PilRC Designer.

Create a Button on the Main Form

We will need to create a button on the Main Form from the design new of the Visual Resource Designer.

1. Open the Object Tree popup list and select the MainForm resource object.
2. To add a button to the form, you will need to select the BUTTON control from the list of controls under the Form category in the Control Palette.
3. Then, move the cursor to the Design View layout screen and position it to be on the form. You should see a "+" symbol appear on the form display.
4. Use your mouse to click once on the form and a new button control should be created.

We are now ready to set the attributes of the button resource by editing the value fields in the Property Editor. Notice the name we use for the button ID. The value that we use is a combination of the form name, the text label of the button, and the type of control. Thus, in this case, we use "Main"+"Create"+"Button".

MainCreateButton BUTTON	
Font	(Standard)
Graphic	
Bitmap	
Graphical	<input type="checkbox"/>
SelectedBitmap	
Misc	
Disabled	<input type="checkbox"/>
ID	MainCreateButton
Text	Create
Usable	<input checked="" type="checkbox"/>
Other	
Anchor	caLeft
Frame	pfbFrame
Position	
Height	12
Left	104
Top	38
Width	45

5. Change the values to:
- | | |
|--------|------------------|
| Font | (Standard) |
| ID | MainCreateButton |
| Text | Create |
| Usable | Checked |
| Anchor | caLeft |
| Frame | pfbFrame |
| Height | 12 |
| Left | 104 |
| Top | 38 |
| Width | 45 |

Figure 4.1: The MainCreateButton property editor window.

Create a New Form

Now we need to make a new form that opens up when we tap that “Create” button we just finished. If you remember how to create a form, do so. If not, go back to Chapter 3 and look at how we created the Main form. Give the new form the name “CreateForm”. Open the Create form for editing. There is no need to change any of the default values, except for the Form Title, which you should set to “Create Errand List.” The completed layout properties for this new form should look like Figure 4.2.

Misc	
ID	CreateForm
Savebehind	<input type="checkbox"/>
Usable	<input checked="" type="checkbox"/>
Other	
DefaultButton	
Frame	pfNoFrame
HelpString	
Menu	
Modal	<input type="checkbox"/>
Title	Create Errand List
Position	
Height	160
Left	0
Top	0
Width	160

Figure 4.2: CreateForm Property Editor window.

Put a Button on the Create Form

The same way you added a button to the Main form, add one from the catalog to the Create form. Give this button the following properties:

Font	(Standard) leave as default
ID	CreateDoneButton
Text	Done
Usable	Checked
Anchor	caLeft
Frame	pfbFrame
Height	12
Left	10
Top	140
Width	36

Compile and Run

Save and close the resource file. Make the project. Make sure POSE is displaying the App Launcher or the Preferences screen and then run the application on your emulator session. It should compile without error, and when you run it, should see the form labeled Errand Manager with a button labeled Create (see Figure 4.3). When you click the Create button, the program should take you to the form labeled Create Errand List, which has a Done button (see Figure 4.4). The Done button will take you back to the first form.



Figure 4.3: The Errand Manager form



Figure 4.4: The Create Errand List form

Take a Look at the Code

There are five files that were modified to make these new resources work this way. We will take a quick look through some of them, and a more in-depth look into others.

ErrandRsc.h

This header file, created by PilRC Designer was updated when you modified and saved the Errand.rsrc file in Constructor. There is now an identifier for the `MainCreateButton` in the group with resource id 1001 (our main form). Also, a new form resource group was created for the `CreateForm` and the `CreateDoneButton` in this file.

Errand.c

If you drop down to the `AppHandleEvent` function, you will see that a new case has been added for the `CreateForm`. Its event handler was set, and that form can be loaded as a result.

Main.c

The `MainFormHandleEvent` has a new case to allow it to respond to `ctlSelectEvents` (like the one that gets sent when we press the Create button). This type of event sends us off to the `MainFormDoCommand` function for further processing. This new function responds to the user selection of the `MainCreateButton`, and uses `FrmGotoForm` to close the `MainForm` and open the `CreateForm`.

Create.c

This new file was added, as mentioned in Chapter 3, to keep all functions related to a certain form in the same file. Since we added the `CreateForm`, we also added the `Create.c` file to handle it. In this form you will see that there are two functions – `CreateFormHandleEvent` and `CreateFormDoButtonCommand`. These are virtual carbon copies of the analogous functions in the `Main.c` file for the `MainForm`. The `CreateFormHandleEvent` function has been set up to handle `frmOpenEvents` (in order to display the form) and `ctlSelectEvents` (to react to the button selections, like the `CreateDoneButton`). In the `ctlSelectEvent` case you can see that `CreateFormDoButtonCommand` is called. This function has a case for the `CreateDoneButton` that uses `FrmGotoForm` to take us back to the `MainForm`.

Errand.h

In the project's header file, three prototypes have been added for the `MainFormDoCommand`, `CreateFormDoButtonCommand`, and the `CreateFormHandleEvent` functions.

An About Form, Menus and Menu Bars

Whenever you create an application, it is helpful to have a screen that gives the user some information about who created it. In the Palm OS world, this is done with an About screen. In this chapter, we will create an About screen and build a menu and menu bar to access it.

- ❖ If you followed Chapter 4, go into the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 5\Src folder and copy the Main.c file to your CodeWarrior Manuals\MW Tutorial for Palm OS\MyErrand\Src folder, overwriting the source file already there.
- ❖ If you skipped Chapter 4, make sure you have a MyErrand folder created in the CodeWarrior Manuals\MW Tutorial for Palm OS folder. Then go to the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 5 folder and copy all the files and folders from that folder into your MyErrand folder.

In this chapter, you will go through the following steps:

- Create an About Form
- Create a Menu and a Menu Bar
- Compile and Run
- Take a Look at the Code

To get started, open the Errand.rcp file in PilRC Designer.

Create an About Form

Create a new form similar to the way you have before, but with one difference. This time create a modal form either by selecting the Modal Form control from the Control Palette or select the regular Form control and check the Modal box. Set its ID to "AboutForm". Select the About form for editing. Set the values to:

ID	AboutForm
Savebehind	Checked
Usable	Checked
Frame	pfNormal
Modal	Checked
Title	About Errand
Height	126
Left	2
Top	32
Width	156

With this form, we have made it Modal. Visually, this means that the form has a border, and generally fills the width of the screen, but not the height. Programmatically, it means that nothing else can be done with the application until the form is dismissed. Even though you will be able to see the `MainForm` in the background, you won't be able to access it. Also, the `Usable` box must be checked in order to set the resource as part of the user interface of the application so it will be drawn. The `Savebehind` field is set if the bits behind the form should be saved when the form is drawn. More information on resource properties for this and other resources can be found in the [PilRC Designer Help](#) file.

With the Control Palette, select the button control and place it on the About form we just created. Give it the following properties:

Font	(Standard) leave as default
ID	AboutOkButton
Text	OK
Usable	Checked
Anchor	caLeft
Frame	pfbFrame
Height	12
Left	62
Top	108
Width	36

Once these have been created, the layout will look like Figure 5.1.



Figure 5.1: The new About form

Create a Menu and a Menu Bar

Now, let's make a menu to take us to the About form we created. Menus are created as part of a Menu Bar, and then the Menu Bar is associated with the appropriate form. This will make more sense as we proceed. First, in the Control Palette, select the MENU resource under the Menus category. This will create a new MENU resource and display it in the Design View layout. The new menu, which is actually a menu bar, should contain two menu items: Application and Help. Normally, we would be done since the Help menu already contains a command that will take you to an About form. However, instead of using this default menu, we will create a new one in order to help you become familiar with manipulating menu groups and commands within the Designer. We are now ready to modify the menu for our purposes. Follow these steps:

1. First, set the ID of the menu resource to "MainFormMenuBar".
2. Select the Application menu group and hit the Delete key or choose Edit->Delete since we will not be needing it.
3. Select the Help menu group and change the Title field to "Options".
4. Within this menu, there should be one menu command, About. Change the Title of this command from "About..." to "About Errand".
5. Change the ID of the menu command from the numerical ID to "OptionsAboutErrand".

Now that we have a new menu bar with an Options menu group, all we have left to do is associate the menu bar with the Main Form. You can do this by selecting the MainForm from the object tree. Then, in the Property Editor for the MainForm, select the Menu field. The Menu field will have a popup list that will have the name of the menu bar we just created. Select the menu bar from the popup list so that the value (MainFormMenuBar) appears in the field. When you have completed these steps, the Design View layout display should look like Figure 5.2.



Figure 5.2: Completed Options menu

Your completed menu bar should look a lot like the menu in Figure 5.2. Make sure to save the resource file once the changes have been completed

Compile and Run

Return to the IDE after closing the resource file. Make the project. Make sure POSE is displaying the App Launcher or the Preferences screen and then run the application on your emulator session. It should compile without error, and when you run it, you should still see the Errand Manager form with its one button. The difference is, now you have a menu on this form. If you are emulating a device with OS 3.3 or earlier, tap the menu silkscreen button to get the menu to appear as in Figure 5.3. The silkscreen buttons are the four to the left and right of the graffiti area on the device. The menu one is in the

bottom left corner of this area, as depicted in Figure 5.3. If you have a device running OS 3.5 or later, you can tap the title bar of the form (where it says “Errand Manager”) to bring up the menu.

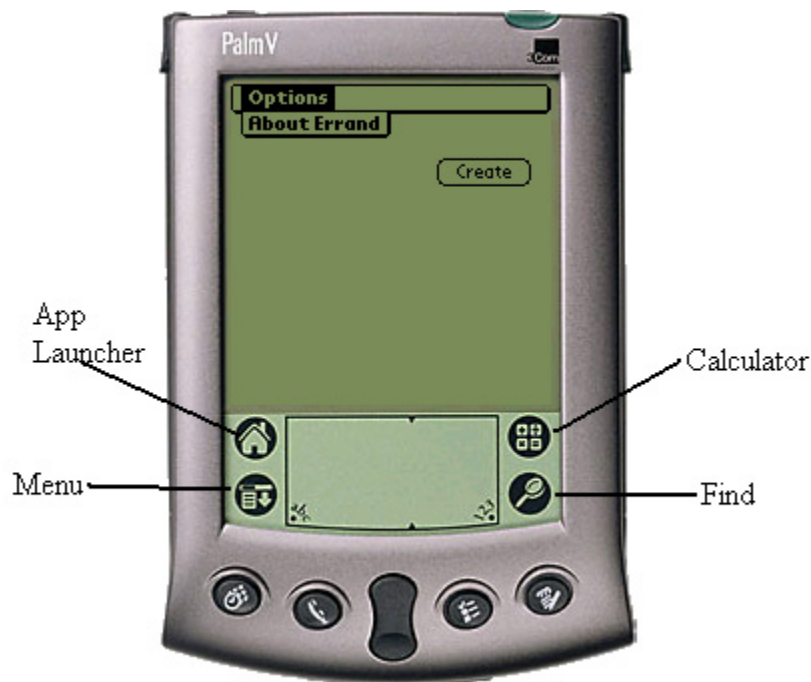


Figure 5.3: The four silkscreen buttons and the menu

If you tap on the “About Errand” menu item, that will bring up the About form we created earlier in the chapter. Since this is a modal form that displays on top of the last non-modal form, you will see the title bar of the Main form in the background, as shown in Figure 5.4. Tapping in that area won’t be recognized since the modal form’s button must be tapped before the user can proceed.



Figure 5.4: The About form displayed with the Main form in the background

Take a Look at the Code

Most of the resources we created in this chapter of the tutorial are managed by the Palm OS without any interaction with the program itself. Here we will look at the files that were modified to accommodate these changes in the program.

ErrandRsc.h

If you open the `ErrandRsc.h` file that was generated by PilRC Designer, you will see the `#defines` for all the new resources we created in this chapter. There is a new form resource for the About form.

Each of these `#define` statements represents components of the `AboutForm`. The first is the form itself, with the second referencing the OK button at the bottom of the form.

A little further down the file, you will see a couple of new resource groups that have been created. The first is the `MENU` resource, which contains the `#define` for the menu bar we built. This resource has two `#defines` – one for the menu title, and one for the one item we created on it.

Main.c

Close `ErrandRsc.h` and open `Main.c` to see the modifications that were made to this file. In the `MainFormHandleEvent` function, a new event has been added to the switch statement. The `MainForm` now handles menuEvents. The same `MainFormDoCommand` function is called that we created for the button presses, but this time we send the menu ID to the function. The menu ID is the ID of the menu item that the user selected.

In the `MainFormDoCommand` function, a new case has been added for the `OptionsAboutErrand` case. This case is entered anytime the user selects the “About Errand” item in the menu. Once we enter this case, we use `MenuEraseStatus (NULL)` to erase the menu display for the current menu. This way there aren’t any drawing problems when we put up the new form. Then we use `FrmInitForm` to initialize the `AboutForm`. We use `FrmDoDialog` to display the form we initialized. With a modal form, `FrmDoDialog` is generally sufficient, since it isn’t necessary to have the extra event handling available to us if we had used `FrmGotoForm`. Using `FrmDoDialog` means we don’t need any event handler for this form. Also, using `FrmDoDialog` leaves the old form there while showing the shorter modal form. Once the OK button is pressed on the form, `FrmDeleteForm` is used to get rid of any leftover traces of the form, and return to the `MainForm` we had displayed earlier. Since we’ve handled everything necessary for this user interaction, we set `handled` to `true` before returning to the event handler for the form and the application.

Labels and a Help String

Since the About screen we created in Chapter 5 is pretty bland, in this chapter we will add some labels and a help string to it.

- ❖ If you followed Chapter 5, there is nothing you need to do – simply continue through the chapter.
- ❖ If you skipped Chapter 5, make sure you have a MyErrand folder created in the CodeWarrior Manuals\MW Tutorial for Palm OS folder. Then go to the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 6 folder and copy all the files and folders from that folder into your MyErrand folder.

In this chapter, you will go through the following steps:

- Add Labels to the About Form
- Add a Help String to the Form
- Compile and Run
- Take a Look at the Code

To get started, open the Errand.rcp file in PilRC Designer.

Add Labels to the About Form

Choose the About form from the object tree to open it for editing. From the Control Palette, select the LABEL control under the Form category. Place a label onto the About form in the Design View layout. Give the label these values:

Font	(Bold)
ID	AboutAppNameLabel
Text	Errand
Usable	Checked
Left	23
Top	29

Drag another label onto the form, and give it the following values:

Font	(Bold)
ID	AboutVersionLabel
Text	Version 1.0
Usable	Checked
Left	10
Top	40

Drag one more label to the form. Give it the following values:

Font	(Bold)
ID	AboutCompanyLabel
Text	© 2001 Metrowerks Corp.
Usable	Checked
Left	13
Top	92

The text for this label is a little different. It will say “© 2001 Metrowerks Corp.” when we’re done with it, but the copyright symbol requires a couple extra steps. From a program called AsciiChart (available as freeware at <http://www.palmgear.com>), we can see that the Hex code for the copyright symbol is A9. You can also use the Character Map utility that is included on most Windows systems under Start->Programs->Accessories->System Tools. Set the Font field in Character Map as “System” and find the copyright symbol. You will notice that Hex code for it is A9 here as well. From Character Map, select the symbol and hit the Select button. It will then appear in the “Character to copy:” field. You can then copy it from here by hitting the Copy button or highlighting it and hitting Ctrl+C.

When you have completed this, your form layout should look like Figure 6.1.



Figure 6.1: Form layout with three labels

Add a Help String to the Form


First, we need to create a string resource. To do this, select the String resource category in the Control Palette. Select the STRING resource control that is located within this category. This will bring up the string resource editor in the Design View layout window.

For this tutorial replace the default string text (“New String”) with the following as the text for this string resource. Make sure to enter the string without hitting any carriage returns, i.e. as a single string on one line. The carriage returns will show as empty lines within the string on the actual device.

“This is the place where you might put information about your product, or where the user might find out more information about you or you might advertise other titles you've put out. Here, we've put our company's web address: `www.metroworks.com`.”

In the Property Editor for the string resource, change the ID field from 1000 to `HelpString1`. The Filename field is there to specify the filename of strings not entered directly in the editor. We can leave this field empty. Save the resource file once you're done.

Now we need to associate the string resource with About Form. Select the About Form from the Object Tree to view the form layout for editing again. In the Property Editor, select the `HelpString` field. Set the value of this field by selecting the string resource we just created (“`HelpString1`”) from the popup list.

If you look at the picture of the form, you will notice a new graphic that appears in the top right corner of the form that looks like this: . When this information icon is tapped by the user, the string that we just created will be displayed.

Compile and Run

Return to the IDE after saving and closing the resource file. It should compile without error, and when you run it, POSE should automatically be launched (because we set the Launch emulator at start of session option in the Palm OS Debugging target settings panel), and you should still see the Errand Manager form with its one button. When you pull up the menu and choose “About Errand”, the new form will look like Figure 6.2.

On the About form, tap the information icon in the top right corner to see the screen in Figure 6.3. This is the Help string, and will always be labeled “Tips”. Tapping the Done button will take you back to the About form.



Figure 6.2: Errand form with labels



Figure 6.3: The Tips form

Take a Look at the Code

None of the resources we created in this chapter required any code changes to work. Instead, the action of accessing the help string and returning to the form is handled completely by the OS. Simply creating the resources and putting the ID references in the right places in Constructor makes them work properly. We will look at the new resources that have been defined by Constructor for this chapter, though.

ErrandRsc.h

Open the ErrandRsc.h file in the editor. Under the resource group there are three new items at the bottom of the group:

```
#define AboutAppNameLabel 1005
#define AboutVersionLabel 1006
#define AboutCompanyLabel 1007
```

These are for the three labels we placed on the form. It's ok if the actual resource ID values are different from the ones shown here just as long as they are different from each other.

The last group in this file is also new. It was created for the string resource, which is the help string we added for display when the information icon was tapped (`HelpString1`). A combination of the context the resource was created in (as a Help resource) is used with the resource type (String) to create a unique name for the identifier. Since we don't reference this resource anywhere in our code, it wasn't necessary for us to create an identifier that was easy to recognize and reference in code.

Adding Bitmaps to the About Form

Bitmaps get their own chapter, since bitmap creation can be a little tricky.

- ❖ If you followed Chapter 6, there is only one thing that you need to do – copy the bitmap images from the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 7\Rsc folder to your MyErrand\Rsc folder.
- ❖ If you skipped Chapter 6, make sure you have a MyErrand folder created in the CodeWarrior Manuals\MW Tutorial for Palm OS folder. Then go to the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 7 folder and copy all the files and folders from that folder into your MyErrand folder.

In this chapter, you will go through the following steps:

- Learn About Bitmap Families
- Add Bitmaps to the About Form
- Compile and Run
- Take a Look at the Code

To get started, open the Errand.rcp file in PilRC Designer.

Learn About Bitmap Families

Because there are so many devices supporting all levels of color from black and white up through 16-bit color, Palm created bitmap families to make the developer's life easier. A bitmap family is a holding place for bitmaps of various depths. Within a bitmap family, you can have one bitmap in black and white, the same bitmap in 2-bit grayscale, another version in 4-bit grayscale, one in 8-bit color and one in 16-bit color. See Figure 7.1 for an example of one bitmap taken through the various depth levels. Then, instead of the developer having to program which color depth to display, the OS will select the highest-level bitmap it can display on the current device.

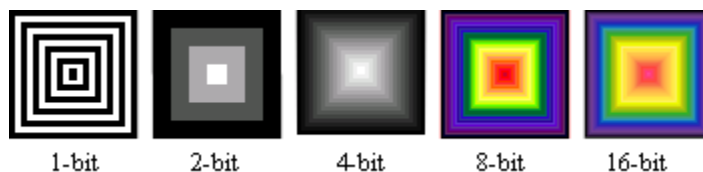


Figure 7.1: An array of bitmaps

At minimum, you will normally need to have a black and white image in your family, unless you intend to only support devices with a higher display level. This might be the case with a custom internal application that will only run on certain devices, or if you've chosen to support a limited feature set for your application. Most of the time you will want a black and white image as well as a color image in your family, so that your


application looks good on color devices, instead of just displaying the same black and white image on color and monochrome devices.

Add Bitmaps to the About Form

We will start out by creating the bitmap resources within the Designer. Note that PilRC Designer does not have an integrated bitmap editor such as Constructor for Palm OS. Therefore, the actual bitmap images will have to be created in a separate bitmap editor such as MS Paint. The bitmap images that you will be using have already been made for you. Therefore, you will only have to associate them with bitmap resources types within the Designer. Follow these steps in order to do this:

1. From the Control Palette, select the BITMAP control from the Images category. This control represents the 1-bit B&W bitmap resource that we will be using.
2. Set the following properties for the resource in the bitmap Property Editor:

ID	AboutMWNameBitmap
Compression	cpNoCompress (default)

3. For the Filename field, select the open button () and navigate to your MyErrnad\Rsc folder. Select the BitmapName.bmp file and hit the Open button. The Property Editor should look like Figure 7.2:

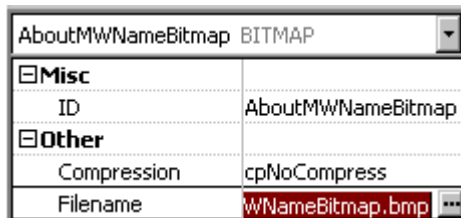


Figure 7.2: Property Editor window for the AboutMWNameBitmap bitmap resource

This should also cause the bitmap image to appear in the Design View.

Create another bitmap resource by repeating Step 1 above, but set the following properties for the newly created resource:

ID	AboutMWLogo1Bitmap
Compression	cpNoCompress (default)

Set the Filename field to the AboutMWLogo1Bitmap.bmp file that is located in the same folder as the first bitmap image.

Create one more bitmap resource. This time, select the BITMAPCOLOR control (for 8-bit 256 color bitmaps) from the Control Palette. You will notice that the Property Editor for this bitmap resource is different from the regular BITMAP resource. It includes fields for picking different transparent colors, and it also has a field for including a bitmap color table. However, the inclusion of a color table is not recommended since it slows down system performance. We will not be using any of these new fields. Therefore, set the Filename to "AboutMWLogo8Bitmap.bmp", and only set the following properties while leaving the others as default:

ID	AboutMWLogo8Bitmap
Compression	cpNoCompress (default)

Notice that AboutMWLogo1Bitmap and AboutMWLogo8Bitmap are different versions of the same bitmap – one is a 1-bit B&W image while the other is an 8-bit color image. Since these bitmaps are really the same image at different pixel depths, we can create a bitmap family resource for this type of image and associate these two bitmaps with it.

First, create a new bitmap family resource by selecting BITMAPFAMILY from the Images category in the Control Palette. In the Property Editor for the bitmap family, notice that there are fields that correspond to different bitmap resolutions. For example the “Bitmap1bpp” field is for 1-bit bitmaps while “Bitmap8bpp” is for an 8-bit version of the bitmap. Therefore, for the Bitmap1bpp field, select AboutMWLogo1Bitmap.bmp and for Bitmap8bpp, select AboutMWLogo8Bitmap.bmp. The selected bitmaps should appear in the Design View window. Finally, set the ID field to AboutMWLogoBitmapFamily and save the resource file.

We are now ready to place the new bitmap resources we have just created onto the About Form. Follow these steps:

1. Open the About form for editing by selecting it from the Object Tree.
2. Select the FORMBITMAP resource from the Forms category in the Control Palette.
3. Place this new resource onto the About form just above the company label by clicking once on the form layout. This should place an empty box on the form where you clicked.
4. The Property Editor for the new form bitmap should appear in the Property Explorer window. If it doesn’t click on the form bitmap (the empty box we just created) so that the Property Editor for it appears.
5. For the Bitmap field, select the first Bitmap resource (AboutMWNameBitmap) we created earlier from the popup list.
6. Set the Left origin field to 5 and the Top to 72.
7. Repeat Step 2 and place the new form bitmap just above the one we created previously and to the right of the App name and version labels.
8. In the Property Editor for the new form bitmap resource, set the Bitmap field to the Bitmap Family resource we created earlier (AboutMWLogoBitmapFamily).
9. Set the Left field to 82 and the Top to 17.
10. Save the resource file.

When all these steps are completed, your form should look like the one in Figure 7.3.

Tip: PilRC Designer also allows you to create high-density bitmap resources. Note the BITMAPFAMILYEX control in the Control Palette. We will not be covering high-density bitmaps in this tutorial, so please refer to the [PilRC Designer Help](#) for more information.



The layout displays the color logo, but remember that the actual image that gets displayed will depend on the highest resolution supported by the system on which the application is running.

Figure 7.3: The completed About form

Compile and Run

Return to the IDE after saving and closing the resource file. Make the project. It should compile without error. Make sure POSE is displaying the App Launcher or the Preferences screen and then run the application on your emulator session. When you run it and choose the “About Errand” menu item you should see the About form with both of the new bitmaps. Depending on whether you run the application on a black and white or color emulated device, you will see Figure 7.4 or 7.5.



Figure 7.4: About form displayed on a black and white device



Figure 7.5: About form displayed on a color device

Take a Look at the Code

No code changes were made to cause the bitmaps to be displayed, nor for the color device to display the higher-depth version of the bitmap. Let's look at the new resources that were added to the Constructor-generated header file.

ErrandRsc.h

Open the ErrandRsc.h file in the editor. In the resource group listings, there are the following new entries, where <ID> represents the ID value that was assigned to each by the Designer:

```
#define AboutMWNameBitmap      <ID>
#define AboutMWLogo1Bitmap    <ID>
#define AboutMWLogo8Bitmap    <ID>
#define AboutMWLogoBitmapFamily <ID>
```

Three identifiers are for the bitmaps we created and the fourth is for the bitmap family. The names come from the names we gave the form bitmap resources, which is a combination of the form they are used in (About), a description of the image itself (MWName or MWLogo), and the resource type (Bitmap or BitmapFamily).

Add a Field to a Form

It's time to add a textfield to the application. Using a field, we will start to see the application come together with data being entered. As before, get the sources first to get started.

- ❖ If you followed Chapter 7, go into the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 8\Src folder and copy the Errand.c, Errand.h, and Create.c files to your CodeWarrior Manuals\MW Tutorial for Palm OS\MyErrand\Src folder, overwriting the source files already there.
- ❖ If you skipped Chapter 7, make sure you have a folder named MyErrand in the CodeWarrior Manuals\MW Tutorial for Palm OS folder. Then copy all the files and folders in the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 8 folder to your MyErrand folder.

In this chapter you will go through the following steps:

- Add a Field and a GSI
- Create an Edit Menu
- Compile and Run
- Take a Look at the Code

To get started, open the Errand.rcp file in PilRC Designer.

Add a Field and a GSI

Select the Create form for editing, and place a FIELD control from the Control Palette onto the form. Give the field the following properties:

Font	(Standard)
Disabled	Unchecked
ID	CreateOtherField
Usable	Checked
Align	pAlignLeft
AutoShift	Unchecked
DynamicSize	Checked
HasScrollBar	Unchecked
MaxChars	256
Numeric	Unchecked
SingleLine	Unchecked
Underlined	Checked
Height	75
Left	10
Top	58
Width	135

Then, anytime you have an editable field on a form, you must have a Graffiti State Indicator (GSI) on the form as well. Select the GRAFFITISTATEINDICATOR control from the Form category in the Control Palette and place it onto the form. Give it the following values:

Left Origin	142
Top Origin	145

Note that the GSI is not treated as a separate resource element. Instead it is part of the CreateForm. You can actually see the reference to it by looking at the Source View of the resource file. From the Source View, select CreateForm from the Object Tree in order to find the source text for the Create Form. In the Create Form source, you should see the line: "GRAFFITISTATEINDICATOR AT (142 145)".

Once the pieces are in place, your layout should look like Figure 8.1. Although the field lines do not appear in the Design View layout, they will appear on the actual device or the Palm OS Emulator.

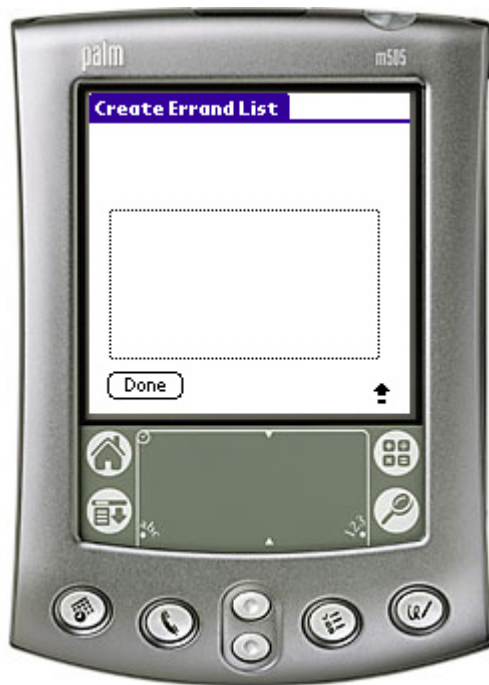


Figure 8.1: A field with a GSI

Create an Edit Menu

Finally, since there is a field on the form, it is important to have an Edit menu for various editing commands. Create a new Menu and change the default name "Application" to "Edit". Set the ID for the menu resource as "CreateFormMenuBar". Open the Edit menu for editing by clicking on it. We can now start adding new menu items to this menu.

1. Since we already have one command in the menu, all we need to do is change its Title from "Exit" to "Cut". Change the AccelChar character field to "X". The AccelChar field represents the shortcut graffiti character for this item. Change the ID field value to "EditCut".

2. Add a new menu item by first clicking on the Edit menu group to select it and then, selecting the MENUITEM control from the Menus category in the Control Palette.
3. Select the new menu item you just created and set the ID to "EditCopy", the Title to "Copy", and the AccelChar to "C".
4. Add another new menu item and set its ID to "EditPaste", the Title to "Paste", and the AccelChar to "P". Unlike the other shortcut characters, the one for pasting is not "V" like you might expect. Palm decided that it was better to break with convention in this case, since Graffiti "V's" are often misentered as "U's", causing unnecessary confusion.
5. Add another new menu item and set its ID to "EditUndo", the Title to "Undo", and AccelChar to "U".
6. Add one more new menu item and set its ID to "EditSelectAll", Title to "Select All", AccelChar to "S".
7. Save the resource file. Since we do not need the Help menu, so select it and hit the Delete key or choose Edit -> Delete. The finished menu group should look like Figure 8.2 in the Design View layout.



Figure 8.2: Edit menu on the Create Form menu bar

Since new menu items can only be added to the end of the menu group, the Designer allows you to rearrange the items through the Design View layout. You can do this by simply dragging and dropping the menu item you wish to move to the location you would like to see it on the menu.

To associate this menu with the Create Form, first choose the Create Form from the Object Tree. Open the Property Editor for the Create Form by clicking on the top portion of the Form or on the blue-colored Form title. In the Menu field, select CreateFormMenuBar from the popup list of menu resources. Save and close the resource file once you are done.

Compile and Run

Open and make the project in the IDE. Make sure POSE is displaying the App Launcher or the Preferences screen and then run the application on your emulator session. When you click on the Create button, it will now take you to the updated create form with the field and the new menu. You can enter text in the field with the keyboard, or use the mouse to create Graffiti strokes. If you use the mouse to create the Graffiti shift stroke (draw a line from bottom to top of the Graffiti area), you will see the GSI react and show up appropriately. It will also display caps lock mode if you draw shift-shift in the Graffiti area to create a locked shift situation.

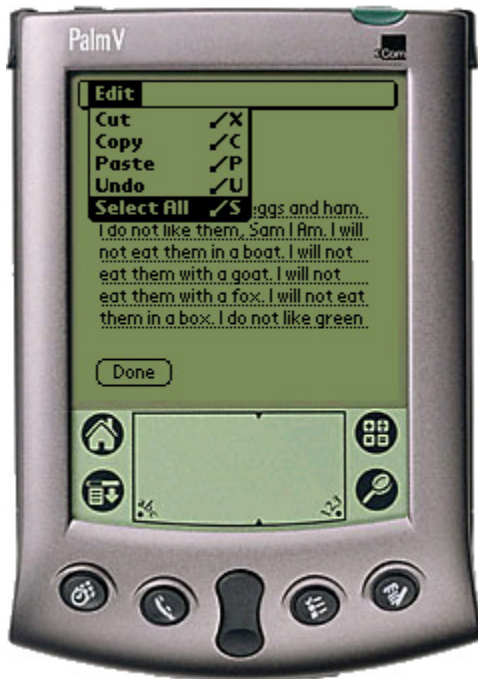


Figure 8.3: Choosing Select All on the field

You can use the menu items or their shortcut characters to modify the text in the field (see Figure 8.3). Experiment with all the options – Cut, Copy, Paste, Undo, and Select All. Try using the menu command by selecting it from the menu and by using the shortcut key. To use the shortcut characters, enter the shortcut Graffiti stroke (a diagonal line from the bottom left to the top right in the Graffiti area) followed by the letter listed in the menu.

Take a Look at the Code

While the Graffiti Shift Indicator is managed by the OS automatically, the field editing menu items must be coded to have the desired effect. In the following paragraphs we will look at the way the code was modified to give the appropriate functionality.

ErrandRsc.h

Open ErrandRsc.h in the editor. There are a number of new resources in the list:

```
#define CreateOtherField <ID> // for the field
#define CreateFormMenuBar <ID> // for the menu bar
#define EditCut <ID> // cut menu item
#define EditCopy <ID> // copy menu item
#define EditPaste <ID> // paste menu item
#define EditUndo <ID> // undo menu item
#define EditSelectAll <ID> // select all menu item
```

When you have finished looking at the resources, close this source file.

Errand.c

Open the Errand.c file, and move to the `GetObjectPtr` function. This is a new function that was added to this file. The code calls the function with an object identifier (the name set up in the ErrandRsc.h file) for a particular resource in the current form. The function then gets a pointer to the current form using `FrmGetActiveForm`. The index of the resource is found using this form pointer and the call `FrmGetObjectIndex`. The index can then be used with `FrmGetObjectPtr` to get a pointer to the object in question.

The index is different from the ID in the following manner. The ID is the number that Constructor generates (we have generally accepted the default value that Constructor creates). It is referenced with the `#define` calls in ErrandRsc.h. On a particular form, all of the objects on it are numbered with an index starting with 0 and going up. The index

is necessary to get a pointer to a particular object, while the ID is much easier to reference and work with. This function makes it seamless to get from the object ID through the index stage and to an object pointer.

Create.c

Open the Create.c file in the editor. In the CreateFormHandleEvent function, there is a new case for handling menuEvents. When a menuEvent is triggered for the Create form, the code calls a new function named CreateFormDoMenuCommand.

Moving to the beginning of this function, there are four local variables defined:

```
Boolean handled = false;
FieldType* fldP = GetObjectPtr(CreateOtherField);
UInt16 startPos = 0;
UInt16 endPos = FldGetTextLength(fldP);
```

The variable handled is initialized to false, and will be set to true if a menu command is selected that is handled in the case statements. The fldP variable uses the GetObjectPtr function to get a pointer to the field object on the form. The last two variables, startPos and endPos, are used to determine how much of the field has text entered in it. The call FldGetTextLength returns the total text length, which is used as the end of the inputted text.

To start the function, there is a call to MenuEraseStatus to delete the current menu from view. Then the case statements begin for the different menu items. The API calls are very straightforward for editing text in fields. For Cut, the appropriate call is FldCut; for Copy, use FldCopy; for Paste use FldPaste; for Undo use FldUndo; and for Select All use FldSetSelection with the starting and ending positions we set with the local variable declarations.

Errand.h

Open Errand.h in the editor. Two new function prototypes were added for the two new functions GetObjectPtr and CreateFormDoMenuCommand. No other changes were made to this file.

A List, a Button, and an Alert

Now we will create a list, button and an alert to facilitate text being added to the text field on the Create form. As before, get the sources first to get started.

- ❖ If you followed Chapter 8, go into the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 9\Src folder and copy the Create.c file to your CodeWarrior Manuals\MW Tutorial for Palm OS\MyErrand\Src folder, overwriting the source files already there.
- ❖ If you skipped Chapter 8, make sure you have a folder named MyErrand in the CodeWarrior Manuals\MW Tutorial for Palm OS folder. Then copy all the files and folders in the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 9 folder to your MyErrand folder.

In this chapter you will go through the following steps:


- Create and Populate a List
- Add a Button and an Alert
- Compile and Run
- Take a Look at the Code

To get started, open the Errand.rcp file in PilRC Designer.

Create and Populate a List

Select the Create form from the Objects list to open it for editing. Select a LIST control from the Form category of the Control Palette and give it the following properties:

ID	CreateErrandsList
Usable	Checked
Visible Items	3
Height	35
Left	12
Top	20
Width	90

To add items to the list, select the Items field in the Property Editor for the List resource. Click the open button () next to "(TStrings)" to open the String Editor window. In the text entry area, replace "One" with "Get oil changed". Replace "Two" with "Go to hardware store" and "Three" with "Go to drugstore". Add six more list items, for a total of seven list items with the following item texts:

Pick up kids
Go to grocery store
Get a haircut
Drop off dry cleaning

Hit the OK button to close the window. When completed, the layout should look like the one in Figure 9.1. Only three items appear in the list since we set `VisibleItems` to 3. The system will automatically handle scrolling the list for viewing the rest of the items.



Figure 9.1: Create Form layout with List of strings

Add a Button and an Alert

Now, select a `BUTTON` control from the Form category of the Control Palette and place it onto the Create Form. Give it these values:

ID	CreateAddtoListButton
Title	Add to List
Anchor	caLeft
Frame	pfbFrame
Height	12
Left	106
Top	39
Width	50
Font	(Standard)
Usable	Checked

Once the button has been added, save the resource file. Create a new Alert resource by selecting the `ALERT` control from the Alert category of the Control Palette. Give it the following properties:

ID	ChooseErrandAlert
AlertType	ppError
Title	Choose Errand

ChooseErrandAlert properties continued:

Message	You must select an errand from the list before choosing Add to List.
Buttons	Leave a single OK button (refer to Ch. 3 if you need a reminder on how to do this)

The completed alert window will look like Figure 9.2.

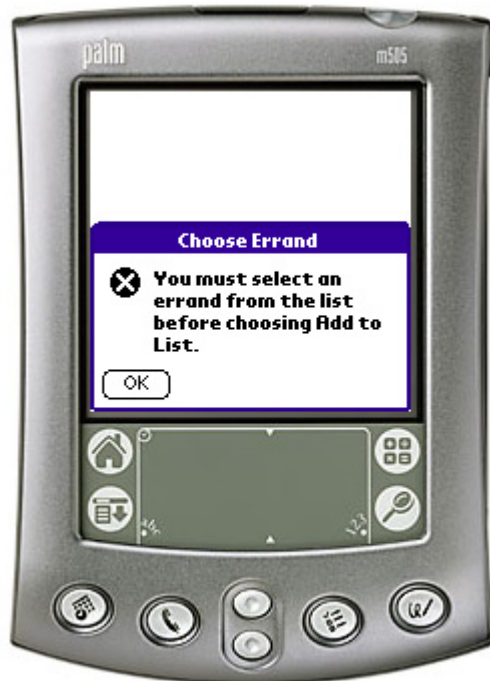


Figure 9.2: The Choose Errand alert

Once the resources have all been entered, save and close the resource file before returning to the IDE.

Compile and Run

Make the project. If you get a compilation error for the resource file referring to the ALERT resource, open Errand.rcp and find the reference to the ALERT resource in question, or you could simply double-click the error in the Errors & Warnings window in the IDE. Try moving the entire ALERT resource text entry (from ALERT ID to END) to the top portion of the file near the other Alert(s) resources that you have created. The error is due to inconsistent line endings that can be fixed by doing this. Make sure POSE is displaying the App Launcher or the Preferences screen and then run the application on your emulator session. The list and button work together to display text from the list in the field. The alert is there to let the user know when they have done something incorrectly. Before selecting any item in the list, tap the "Add to List" button to see the alert pop up as in Figure 9.3. Select an item in the list (use the scrollbars in the list to move to other items that are not visible) and press the "Add to List" button. You should

see the item from the list appear in the field, and the cursor should be on the next line of the field, as shown in Figure 9.4.

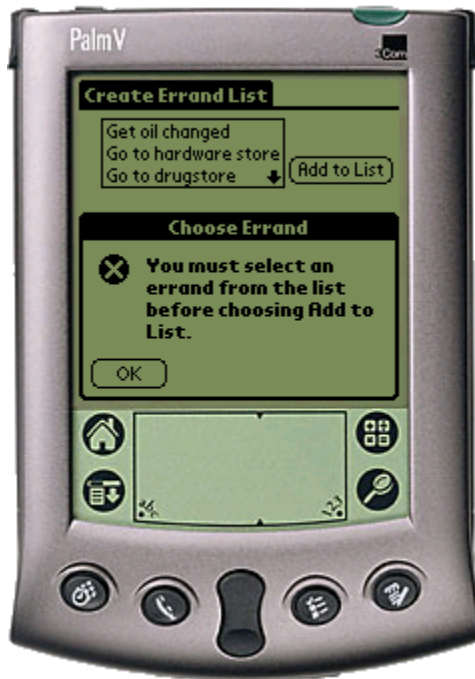


Figure 9.3: The no list selection alert

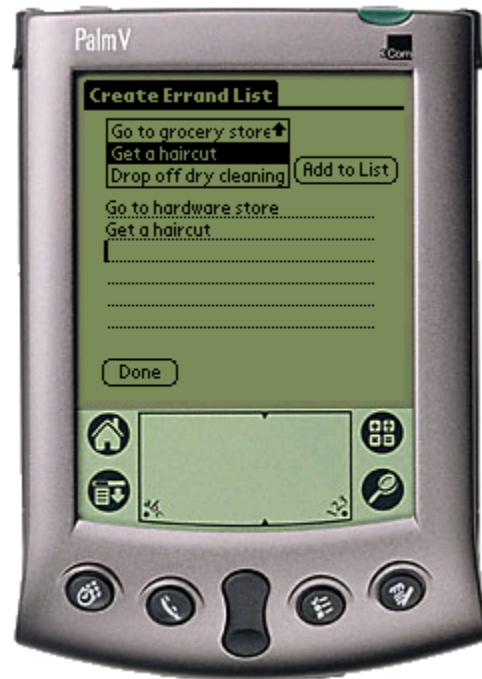


Figure 9.4: Text added to the field

Take a Look at the Code

Here we will look into the code that was created to manipulate the text from the list into the field.

ErrandRsc.h

Open ErrandRsc.h in the editor. There are two new values in the resource group – the CreateAddtoListButton and the CreateErrandsList. These reference the list and button we added to the CreateForm. A new Alert resource was created for the ChooseErrandAlert as well.

Create.c

Open the Create.c file in the editor. The CreateFormHandleEvent has a new piece in it. In the frmOpenEvent, there is a call to LstSetSelection to have the CreateErrandsList show no items as selected upon drawing the form. This default is more useful in this case than having the first item selected all the time. The zero-based parameter for the item to be selected accepts a negative number to choose no selection.

In the CreateFormDoButtonCommand function, there are several new local variables.

```
Char *listTextP, *newline = "\n";
FieldType *fldP = GetObjectPtr(CreateOtherField);
ListType *listP;
```

These handle pointers to text and objects on the form. There is also a new case for the CreateAddtoListButton. In this case, we start with a pointer to the list object. Using LstGetSelection and LstGetSelectionText with that pointer, we are able to get a pointer to the text that the user selected in the CreateErrandsList. Then, as long as

the user selected an item, we use `FldGrabFocus` to set the focus/insertion point to the field, just in case it isn't there already. Using `FldInsert`, we add the text from the character pointer we got earlier. One of the parameters of `FldInsert` is the length of the string, and `StrLen` returns that for our use here. If the user didn't select anything, then `listTextP` will be `NULL`, and we go into the else case where `FrmAlert` is called to display the `ChooseErrandAlert`. Since everything for this button was handled in this function, `handled` is set to `true`, and returned to the form handler.

Popup Trigger and Saving to a Database

In this chapter we will make a popup trigger and a button to save some information to a database. As before, get the sources first to get started.

- ❖ If you followed Chapter 9, go into the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 10\Src folder and copy the Errand.h, Errand.c and Create.c files to your CodeWarrior Manuals\MW Tutorial for Palm OS\MyErrand\Src folder, overwriting the source files already there.
- ❖ If you skipped Chapter 9, make sure you have a folder named MyErrand in the CodeWarrior Manuals\MW Tutorial for Palm OS folder. Then copy all the files and folders in the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 10 folder to your MyErrand folder.

In this chapter, you will go through the following steps:

- Create a Popup Trigger
- Add a Button and Two Alerts
- Compile and Run
- Take a Look at the Code

To get started, open the Errand.rcp file in PilRC Designer.

Create a Popup Trigger

Open the Create form for editing. From the Control Palette, select the POPUPTRIGGER control under the Form category and place it onto the Create form just above the Add to List button. Give it the following properties:

Font	(Standard)
ID	CreateDatePopTrigger
Text	Choose
Usable	Checked
Anchor	caLeft
Height	12
Left	105
Top	20
Width	50

Leave the other properties as default and save the resource file.

A popup trigger needs a list to go with it. The list gets displayed when the popup trigger is selected on the form. We will create the list first, then go back and associate it with the popup trigger we created earlier. Add a new list object to the form by selecting the LIST control from the Form category of the Control Palette and placing it onto the Create form near the popup trigger you just created. Set it up with the following values:

ID	CreateDateList
Usable	Unchecked
VisibleItems	7
Left	119
Top	20
Width	30

Set (TStrngs) to the following List Items in the String Editor window:

Sun
Mon
Tues
Wed
Thurs
Fri
Sat

In order to associate the list with the popup trigger, we must create a new resource called a popup list. A popup list resource is used to combine a list and a popup trigger together in order to create a dropdown list.

Select the POPUPLIST control from the Form category of the Control Palette and place it onto the Create form. It does not matter where you try to place it on the form, the popup list will always initially appear in the upper left corner of the form layout. However, it does not matter where on the form it is located because the popup list resource is only displayed in the Design View layout. Therefore, it will not appear on the actual device during runtime. The popup list resource only has two property fields that need to be set, one for the list and one for the popup trigger. Set the List field to CreateDateList and the PopupTrigger field to CreateDatePopTrigger. Save the resource file when you are done.

Tip: PilRC Designer offers a “Trigger and List” template control that will create a List, a popup trigger, and a popup list all at once. Although the individual properties of each resource must still be set, using the template can save you time as opposed to having to select and place each resource one by one. This template is located under the Template category of the Control Palette.

When all the resources are entered, the form will look cluttered, as in Figure 10.1. This won’t be the case when the application is run, since the list that is associated with the popup won’t be visible unless the trigger is selected and the popup list resource will not appear at all.

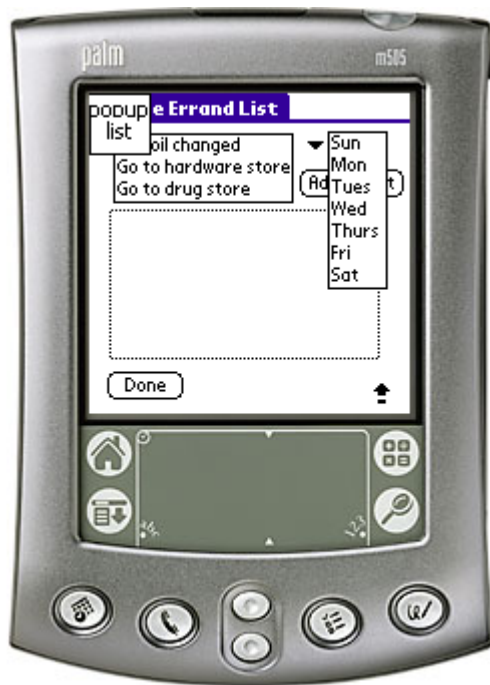


Figure 10.1: A cluttered Create form

Add a Button and Two Alerts

Select a button from the Control Palette Form category and place it on the Create form near the Done button. Give the new button the following properties:

ID	CreateSaveDayButton
Title	Save Day
Anchor	caLeft
Frame	pfbFrame
Height	12
Left	90
Top	140
Width	45
Font	(Standard)
Usable	Checked

Save the resource file. Create a new alert resource. Select it for editing and set its values to:

ID	NeedMoreInfoAlert
AlertType	ppError
Title	Need More Info
Message	You must select a day from the list and fill in errands in the field before saving the day to the database.
Buttons	Leave a single OK button (refer to Ch. 3 if you need a reminder on how to do this)

Save this alert resource and create a new one. Select it for editing and give it the following properties:

ID	SavedAlert
AlertType	ppInformation
Title	Saved!
Message	^1's errands have been saved to the database.
Buttons	Leave a single OK button.

The ^1 will be replaced by the appropriate day identifier during runtime. Save the resource file before returning to the IDE.

Compile and Run

Make the project. Make sure POSE is displaying the App Launcher or the Preferences screen and then run the application on your emulator session. When you click the create button, it will look like Figure 10.3. Experiment with the popup trigger and adding list items to the field and clicking the Save Day button. If you haven't selected the day and entered text in the field, then you should see the alert in Figure 10.4.

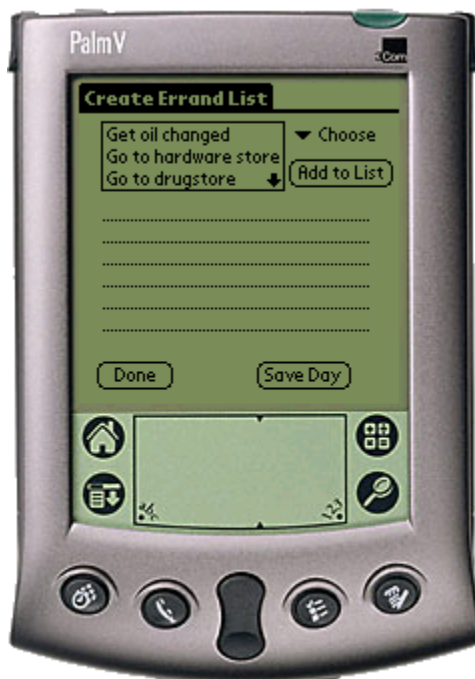


Figure 10.2: The full Create form with all the resources on it

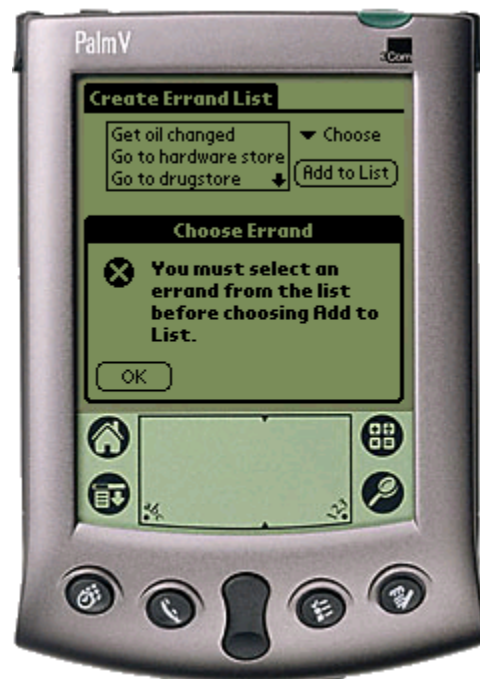


Figure 10.3: Trying to save without all the information entered

If you have selected a day from the popup trigger and have entered text in the field, click on Save Day to save the information in the database. You should see the screen in Figure 10.4 with whichever day you set in the popup trigger. Continue to experiment with picking different days out of the popup list and saving various things to the database. Exit the app, go back into the Create form and select a day that you know you saved, and see it display the text you saved in the previous session.

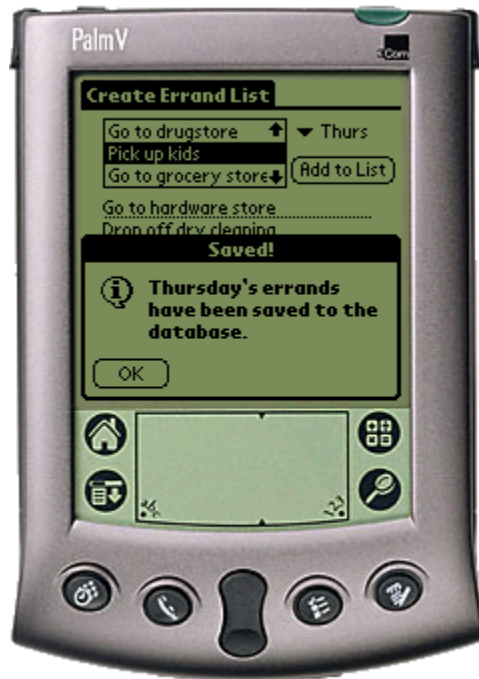


Figure 10.4: Result of saving the day

Take a Look at the Code

There is a pretty significant amount of code that was added in this chapter to create the database where the text is saved. In this section, we will discuss those additions.

ErrandRsc.h

Open ErrandRsc.h in the editor. There are three new identifiers:

```
#define CreateSaveDayButton <ID>
#define CreateDateList <ID>
#define CreateDatePopTrigger <ID>
```

The first is the button we added to the form. The second and third refer to the list and the popup trigger, respectively. Then there are two new resource constant definitions for the two alerts we created. These are the `NeedMoreInfoAlert` and the `SavedAlert`. When you have seen the additional resource defines, close the file.

Errand.c

Open the Errand.c file in the editor. A couple of functions have been modified to accommodate the new database functionality. A few `#defines` have also been added at the top of the file:

```
#define kCreator 'ErrD'
#define kDBType 'DATA'
#define kDBName "ListErrD"
```

These defines will be used to create and open the database. By using the same creator code for the database (when we create it) that was used to create the application, we can ensure that when our application is deleted from the device, the database will also be deleted. Since our application is the only application on the device that is using this database, this is the right way to handle it. It also means that when you view the size of the application on the device, the size will be the sum of the application size and the database size. The database type is `'DATA'` to distinguish it from a resource database

and from an application. The database name must be unique. The convention from the developer community is that this should be done by using a unique identifier (in case you have several databases in the app), followed by the application's creator code. Since the creator code must be registered with Palm, this value is unique. Then, the additional identifier for databases within an application ensures that no two databases on any device will have the exact same name.

Also, a `DmOpenRef` global of `gDB` has been defined. This variable holds a reference to the database, once it has been opened. This variable will allow us to do many things with our database once we have it.

The `AppStart` function takes care of creating the database if it doesn't exist or opening one that does. The first time the application is run on a new device (or fresh emulator session), there won't be any database, so it will have to be created. After that, it is important to reopen the one that exists on the device, so the user sees the same data they saved the last time they used the application. First, we attempt to open the database, in case it already exists on the device. This code uses the `DmOpenDatabaseByTypeCreator` call, and references the database type, creator code of the application and a mode (`dmModeReadWrite`) through constants. If the database exists, a reference value will be saved in the `gDB` global variable. If it doesn't exist, then we call `DmCreateDatabase` to create that database. Then, since we just created it, the database should now exist, and we can open it with `DmOpenDatabaseByTypeCreator` again. If a reference value still isn't stored in the `gDB` variable, then there was some kind of problem opening the database, and the application is exited.

In the `AppStop` function, there is a new call to `DmCloseDatabase` to close the database. The global `gDB` is the variable we use to tell the program which database to close. This way we make sure to clean up memory and make sure everything is closed before exiting the application.

Create.c

In the `CreateFormHandleEvent` function, there is a new `LstSetSelection` call to set the popup list to have no selection until the user actually taps on the trigger and picks something. There is also a new case for `popSelectEvents`. This case calls the new function `CreateFormDoTriggerCommand` with the ID of the trigger that was selected.

The `CreateFormDoButtonCommand` function has several new local variables to use with the new case for the `CreateSaveDayButton`:

```
Err error;
Int16 day;
ErrandDB record;
Char *dayList[7] = {"Sunday", "Monday", "Tuesday",
    "Wednesday", "Thursday", "Friday", "Saturday"};
```

`ErrandDB` is a simple struct defined for this application as:

```
typedef struct ErrandDB {
    Weekday day;
    Char errands[257];
} ErrandDB;
```

This struct uses the enumerated data type `Weekday` defined as:

```
typedef enum Weekday{Sun, Mon, Tues, Wed, Thurs, Fri, Sat}
Weekday;
```

In the case for the `CreateSaveDayButton`, we start by getting a pointer to the text in the field using `FldGetTextPtr`. Then, using `LstGetSelection`, we find out which day was chosen in the popup trigger. However, if either value is blank, then there isn't anything to store to the database. It is important to give the user feedback on why nothing will be happening when they press the Save button, so we use `FrmAlert` to display the `NeedMoreInfoAlert`. This alert tells the user that text must be in the field with a day selected in the trigger in order to save anything. If both pieces of information have already been entered, then we need to populate the record variable and save that to the database. The day data is a simple assignment statement, and we use `StrCopy` to copy the string into the struct value. Then we call the new `AddToDatabase` function with the populated record. If the record was safely saved to the database, then we display the `SavedAlert` with `FrmCustomAlert` and a string from the `dayList` array. If you remember, we created the `SavedAlert` resource with a "`^1`" value in the message text. With a custom alert, you can pass up to three custom strings to be displayed in the alert in place of the "`^1`", "`^2`" or "`^3`" values in the text. This means you get the ease of use of an alert with more flexibility in the text. This alert only has one custom value, so we pass `NULL` for the last two parameters. It uses the day that was saved to give a more solid feeling to the user that the right data was saved. Since everything was handled for the button, the `handled` variable is set to `true` and returned to the form handler.

The button code invoked a new function called `AddToDatabase` that is also in this file. This function has a couple of local variables:

```
Err error;
UInt16 recIndex = dmMaxRecordIndex;
MemHandle recH;
MemPtr recP;
UInt16 index;
```

The only one that needs further discussion is `recIndex`, which finds out the number of records in the database with the macro `dmMaxRecordIndex` from the API. The function begins by calling `MemPtrNew` to set aside memory for the `recP` pointer. Then the day passed to this function in the struct pointer is stored in the `recP` pointer and passed to the `GetFromDatabase` function. This is a new function created for this chapter, and we will discuss it in detail in a few moments. We then call `MemPtrFree` to deallocate the memory for this pointer. The `GetFromDatabase` function determines if data already exists for a specified day, and returns the index of that record in the `index` variable. If there is no data in the database for a specified day, then the index will be negative. This allows us to append additional information to an existing record if it exists. If no data is found for the specified we call `DmNewRecord` with `recIndex` as a parameter to add a new record to the end of the database. This call allocates space in the database, so we provide the size of the `Weekday` enumerated piece and the character string together). If the system is able to allocate enough space for the record, a handle is created. Then the handle is locked (it must be locked in order to have information written to it or read from it) with `MemHandleLock`, giving us a pointer to the memory. We use `DmWrite` to write to the memory referenced by the pointer. Then we use `DmReleaseRecord` to free the busy bit, since we aren't using the record any more and `MemHandleUnlock` to unlock the memory chunk. If data is found by the `GetFromDatabase` function, `newSize` is created and set to have the size of the old record plus the size of the added additional errands part of the struct. Then there is a call to `DmResizeRecord` with the index of the record and the new size to change the space allowed for the handle of the existing record. `MemHandleLock` locks the record handle and gives us a pointer to the memory, while `DmWrite` puts the errands data from the struct into the record after the `Weekday` value. `DmReleaseRecord` clears the busy bit, and `MemHandleUnlock` releases the memory allocated. If there any errors in this process, an error code is returned.

The `CreateFormDoTriggerCommand` function has a couple of local variables it uses to process the trigger input. It is sent the ID of the list item that was selected, and defines a few other variables, including `recP`, a pointer to the `ErrandDB` struct. Space for `recP` is allocated with `MemPtrNew`. Assuming the selected day is a positive value (a negative value means that nothing is selected in the list), then it is a value between 0 and 6 (representing the 7 values we entered as list items). This corresponds nicely with the enumerated data type we created to hold the day value. The selected list item is saved as the day component of the struct. We use `FldDelete` to delete anything currently in the field (deleting from 0 to the result of `FldGetMaxChars`, which finds the total number of characters the field can hold). The `UInt16` value `foundData` lets us know if there is already data in the database for that day using the function `GetFromDatabase`. If there is, then `FldInsert` is called to put the character string that was returned from the database as part of the `recP` struct into the field. When we are done with `recP`, we call `MemPtrFree` to deallocate the memory. The variable `handled` is never set to `true` because we want to ensure the OS finishes working with the popup trigger. The OS manages the popping and the hiding of the list after a selection is made. If we were to return `handled = true`, the OS would not handle this event, and the trigger wouldn't function properly.

The last new function is the `GetFromDatabase` function. This function is used when a day is selected from the popup trigger, so the user can see what text has already been entered for the day. A pointer to the `ErrandDB` struct is passed in that has the day filled in. The local variable `numRecs` is initialized by calling `DmNumRecords` and passing the database reference as the parameter. This allows us to determine how many records are currently saved in the database. Then we cycle through the records, calling `DmQueryRecord`, which opens the record for read access only, and gives us a handle. Then we use `MemHandleLock` to lock the handle and give us a pointer to the data. If a record is found in the database with the appropriate day value, the text is copied into `recordP` using `StrCopy`, for use by the calling function. If a value is found, the variable `index` is changed to match the index of the record. When finished, we call `MemHandleUnlock` to release the chunk. The `index` value is returned. If it is still `-1`, then no value was found in the database to match the requested day. If it is a non-negative value, then a matching record was found, and `index` contains the location in the database of the matching record.

Errand.h

In the project's header file, the enumerated data type `Weekday` is defined so it can be used in the struct declaration for `ErrandDB`. `ErrandDBPtr` is defined, as well, as a pointer to the `ErrandDB` struct. The global variable `gDB` (which is defined in `Errand.c`) is declared. There are also prototypes for the new functions `GetFromDatabase`, `AddToDatabase`, and `CreateFormDoTriggerCommand`.

Adding a View Form

In this chapter we will create a new form and add a button the Main Form to take us to that new form. As before, get the sources first to get started.

- ❖ If you followed Chapter 10, go into the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 11\Src folder and copy the Errand.c, Main.c, View.c and Errand.h files to your CodeWarrior Manuals\MW Tutorial for Palm OS\MyErrand\Src folder, overwriting the source files already there.
- ❖ If you skipped Chapter 10, make sure you have a folder named MyErrand in the CodeWarrior Manuals\MW Tutorial for Palm OS folder. Then copy all the files and folders in the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 11 folder to your MyErrand folder.

Open the project, and add the View.c file to the project. Put this file in the Source group.

In this chapter you will go through the following steps:

- Create a Form with Push Buttons
- Add a Field and Two Buttons
- Compile and Run
- Take a Look at the Code

To get started, open the Errand.rcp file in PilRC Designer.

Create a Form with Push Buttons

Create a new form resource. Set its Title to “View Errands” and its ID to ViewForm. Leave all the values at their default. Then, from the Form category of the Control Palette, select a PUSHBUTTON control and place it onto the View form. Give it the following properties:

ID	ViewSunPushButton
Title	Sun
Anchor	caLeft
Frame	pfbFrame
Height	12
Left	27
Top	22
Width	25
Font	(Standard)
Usable	Checked
GroupID	1

Place six more Push Buttons onto the form, and assign their properties based on the following table (all other properties can be left as default):

Title	Mon	Tues	Wed	Thurs	Fri	Sat
Height	12	12	12	12	12	12
Left	53	79	108	37	68	94
Top	22	22	22	35	35	35
Width	25	28	25	30	25	25
Usable	Checked	Checked	Checked	Checked	Checked	Checked
Group ID	1	1	1	1	1	1
Font	Standard	Standard	Standard	Standard	Standard	Standard

Set the ID for each pushbutton using the same format as you used for Sun (i.e. “View”+ Title + “PushButton”). For example, the ID for Mon would be ViewMonPushButton, Tues would be ViewTuesPushButton, and so on. Save the resource file once you are done.

When the push buttons have all been completed, the form will look like Figure 11.1.

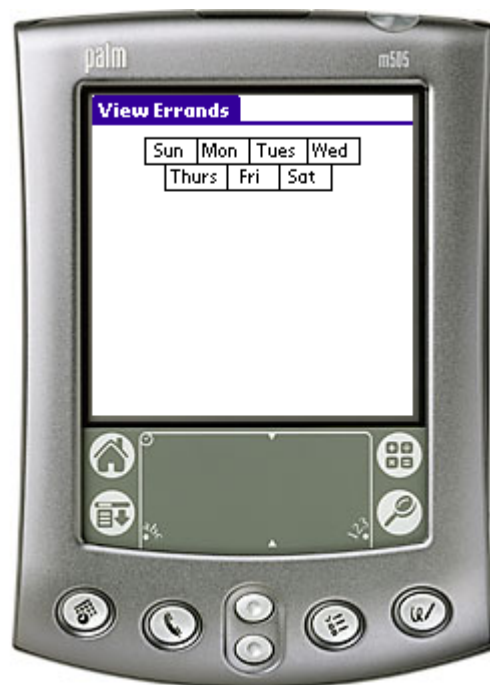


Figure 11.1: Seven push buttons on a form

Note that we have specified the same group ID for all of the push buttons. Controls within the same group are mutually exclusive, like a radio button. Since all seven of the push buttons are in group 1, only one of the seven can be pressed at any given time. If we had assigned no group to the push buttons, all seven could be pressed at the same time. That didn’t make sense for our application, but the requirement that only one button be pressed did, so each of the buttons is in the same group.

Add a Field and Two Buttons

Select a FIELD object from the Control Palette and place it onto the View form. Give it the following properties:

Font	(Standard)
Disabled	Unchecked
ID	ViewErrandField
Usable	Checked
Align	pAlignLeft
AutoShift	Unchecked
DynamicSize	Unchecked
HasScrollBar	Unchecked
MaxChars	256
Numeric	Unchecked
SingleLine	Unchecked
Underlined	Unchecked
Height	79
Left	2
Top	57
Width	158

Then drag a button from the Catalog onto the form. Set the following attributes:

ID	ViewDoneButton
Title	Done
Anchor	caLeft
Frame	pfbFrame
Height	12
Left	6
Top	143
Width	36
Font	(Standard)
Usable	Checked

Save the resource file and select MainForm from the Object Tree to open it for editing. Place a button onto the form, and give it the following settings:

ID	MainViewButton
Title	View
Anchor	caLeft
Frame	pfbFrame
Height	12
Left	104
Top	69
Width	45
Font	(Standard)
Usable	Checked

Save the resource file before returning to the IDE.

Compile and Run

Make the project. Make sure POSE is displaying the App Launcher or the Preferences screen and then run the application on your emulator session. The main form now has a

new button labeled View. When you click the View button, you will be taken to the View form. When you select one of the push buttons, you should see the text “No Errands Specified”, as in Figure 11.2, or the text you saved in earlier sessions in the database from the Create form, as in Figure 11.3. Experiment with clicking on various push buttons. Then go back to the Main form and go to the Create form to modify what is saved in the database. Make sure these changes are reflected in the View form.



Figure 11.2: Viewing a day with no errands saved in the database

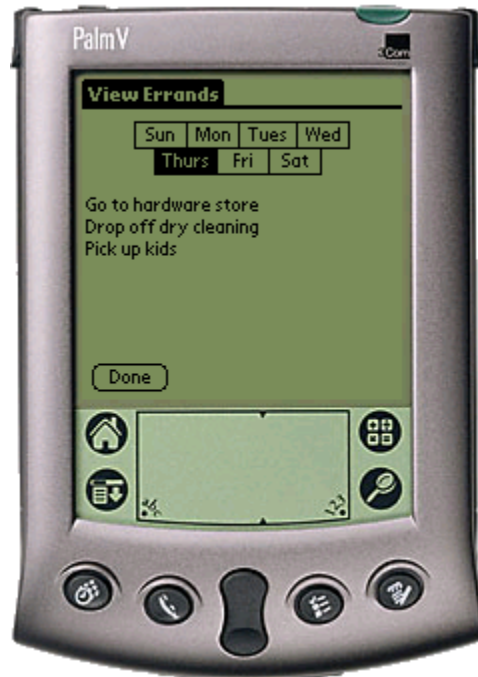


Figure 11.3: Viewing the errands saved for a particular day

Take a Look at the Code

This chapter’s code reuses the data retrieval code written in Chapter 10. There is also a bit of new code that we will discuss in the following paragraphs.

ErrandRsc.h

Open ErrandRsc.h in the editor. There is a new button defined for the MainViewButton. A little further down, there is a new group of resources. The members are the following:

```
#define ViewForm          <ID>
#define ViewDoneButton    <ID>
#define ViewErrandField   <ID>
#define ViewSunPushButton <ID>
#define ViewMonPushButton <ID>
#define ViewTuesPushButton <ID>
#define ViewWedPushButton <ID>
#define ViewThursPushButton <ID>
#define ViewFriPushButton <ID>
#define ViewSatPushButton <ID>
```

The first three definitions are very much the same as they have been in other chapters – one for the form itself, one for a button on the form, and one for the field. Then we have the definitions of each of the push buttons we added and named.

Errand.c

Open `Errand.c` in the editor. In `AppHandleEvent`, there is a new case to set the event handler for the `ViewForm`. This is the only modification necessary in this file.

Main.c

Open `Main.c` in the editor. In the `MainFormDoCommand` function, there is an additional case for the `MainViewButton`. When this button is selected, `FrmGotoForm` is called to change to the `ViewForm`.

View.c

`View.c` is the new file that has been added to control the functionality of the `ViewForm`. Open this file in the editor. The last function in the file is the `ViewFormHandleEvent` function. It is very similar to the event-handling functions for other forms, and has cases for the `frmOpenEvents` and the `ctlSelectEvents`.

In the `ctlSelectEvent`, `ViewFormDoButtonCommand` is called. This function has a case for each of the buttons on the form. The first case is for the `ViewDone` button, and calls `FrmGotoForm` to return to the Main form. The next seven cases are for each of the seven push buttons we created. Each of these cases calls the function `PutTextForDay` with the appropriate day and the `ViewErrandField` as parameters. This function finds the text in the database for the specified day and displays it in the field. Once that is finished, `handled` is set to true so that no further OS action is taken on this event.

The `PutTextForDay` function has several local variables set aside, most of which we've discussed before. There is a character pointer defined for a default string, in case no database record is found for a selected day. We begin the function with `MemPtrNew` to allocate space for the struct and get a pointer to that memory. Then, the first part of the struct is filled with the day that was passed in as a parameter. The field on the form is deleted with a call to `FldDelete` between the first and last spots on the field, so that nothing will be displayed except what we print there. A `UInt16` variable is used to see if the `GetFromDatabase` function (that we discussed earlier in the `Create.c` file) returns any record. If that function finds a matching record, the field information is put in the struct that was passed to it. Then we use `FldInsert` to put the string from the struct into the field on the form that was passed in as a parameter. If no record was found that matched the day in question, `FldInsert` is used with the default string defined in this function. When we are finished with the pointer to the struct, it is important to call `MemPtrFree` to release the memory we allocated.

Errand.h

Open the `Errand.h` file in the editor. At the bottom of the file, you will see that three new function prototypes have been added for the functions in the `View.c` file: `PutTextForDay`, `ViewFormDoButtonCommand`, and `ViewFormHandleEvent`.

A Form for Completion and Deletion

In this chapter, we will be going through the steps to delete a complete errand or set of errands from the database. As before, get the sources first to get started.

- ❖ If you followed Chapter 11, go into the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 12\Src folder and copy the Errand.c, Main.c, Complete.c and Errand.h files to your CodeWarrior Manuals\MW Tutorial for Palm OS\MyErrand\Src folder, overwriting the source files already there.
- ❖ If you skipped Chapter 11, make sure you have a MyErrand folder created in the CodeWarrior Manuals\MW Tutorial for Palm OS folder. Then go to the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 12 folder and copy all the files and folders from that folder into your MyErrand folder.

Open the project, and add the Complete.c file to the project. Put this file in the Source group.

In this chapter, you will go through the following steps:

- Create a Form, Buttons, and an Alert
- Add Several Labels and Checkboxes
- Compile and Run
- Take a Look at the Code

To get started, open the Errand.rcp file in PilRC Designer.

Create a Form, Buttons, and an Alert

Open the Main form for editing. Add a new button to the form with the following values:

ID	MainCompleteButton
Title	Complete
Anchor	caLeft
Frame	pfbFrame
Height	12
Left	104
Top	101
Width	45
Font	(Standard)
Usable	Checked

Save the form and resource file when the button is completed.

Create a new alert and set the following values:

ID	DeletedAlert
AlertType	ppInformation
Title	Delete Successful
Message	All days marked completed have been deleted from the database.
Buttons	Leave a single OK button (refer to Ch. 3 if you need a reminder on how to do this)

When finished, save the alert resource.

Create a new form. Set the Title to “Complete” and the ID to “CompleteForm” (leave all other values at their default values). Select the form for editing. Add two buttons to the form and give them the properties based on the following table:

ID	CompleteDoneButton	CompleteDeleteButton
Title	Done	Delete Checked Items
Anchor	caLeft	caLeft
Frame	pfbFrame	pfbFrame
Height	12	12
Left	6	9
Top	143	143
Width	36	95
Font	(Standard)	(Standard)
Usable	Checked	Checked

Add Several Labels and Checkboxes

Add seven labels to the Complete form. Set their values based on the following table:

Text	Sun	Mon	Tues	Wed	Thurs	Fri	Sat
Left	7	7	6	7	3	10	8
Top	23	39	56	73	89	107	124
Usable	Checked	Checked	Checked	Checked	Checked	Checked	Checked
Font	Standard	Standard	Standard	Standard	Standard	Standard	Standard

Set the ID for each label resource in the following manner: combine the Form name (“Complete”) + the label text (“Sun”, “Mon”, etc.) + the resource type (“Label”). For instance, the ID corresponding to Sun would be “CompleteSunLabel”.

Add a CHECKBOX resource to the Complete form by selecting it from the Form category of the Control Palette. Set it up with the following properties:

ID	CompleteSunCheckbox
Text	Completed
Usable	Checked

Anchor	caLeft
Checked	Unchecked
GroupID	0
Height	12
Left	94
Top	22
Width	65
Group ID	0
Font	(Standard)

Then add six more checkboxes to the form. Give them values based on the following table. Set the ID field the same we did for the Sunday checkbox (e.g. ID for Mon would be "CompleteMonCheckbox"):

	Mon	Tues	Wed	Thurs	Fri	Sat
Left	94	94	94	94	94	94
Top	39	55	73	90	107	123
Width	65	65	65	65	65	65
Height	12	12	12	12	12	12
Usable	checked	checked	checked	checked	checked	checked
Selected	unchecked	unchecked	unchecked	unchecked	unchecked	unchecked
Group ID	0	0	0	0	0	0
Font	Standard	Standard	Standard	Standard	Standard	Standard
Text	Completed	Completed	Completed	Completed	Completed	Completed

When these resources have all been added to the Complete form, it should look like Figure 12.1.

Save your resource file in PilRC Designer before returning to the IDE.



Figure 12.1: Complete form with labels, buttons, and checkboxes

Compile and Run

Make the project. Make sure POSE is displaying the App Launcher or the Preferences screen and then run the application on your emulator session. On the first screen you will see a new button that will take you to the Complete form. Before choosing that button, go to the Create form and make sure there are a couple of days that have information saved for them. Then go to the Complete form, and you will see something like Figure 12.2. Any day that you saved an entry for should be unchecked. Then, to signify that you finished the errand, you could check the box for one of the records, and choose Delete Checked Items to delete it from the database. After clicking the Delete button, you should see a screen like the one in Figure 12.3 to let you know there were no problems deleting the data. Then go to the View form to see if the data was really deleted.

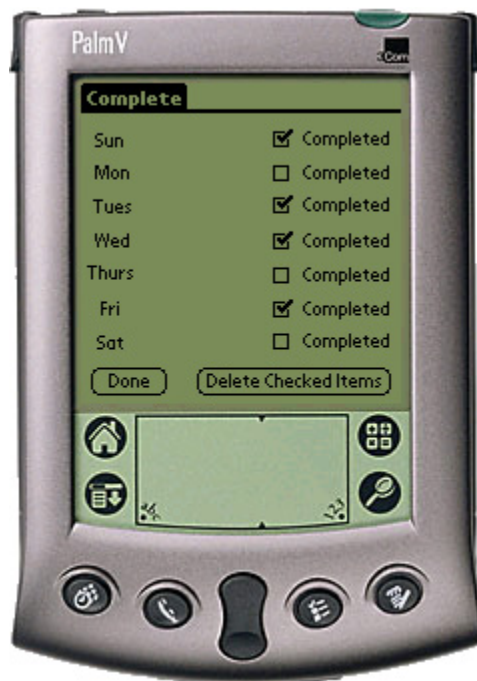


Figure 12.2: Complete form with errands saved for Monday, Thursday and Saturday

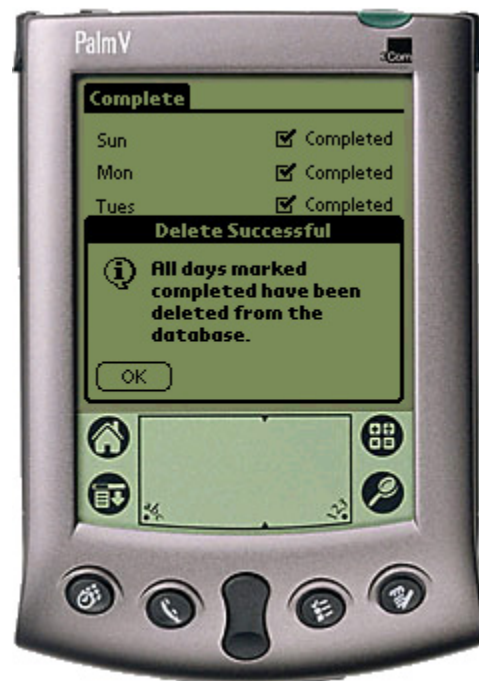


Figure 12.3: Confirmation that the checked records were deleted

Continue to experiment with creating, viewing and completing records in the program. When you are satisfied that you understand how the program functions, go into the next section to see what code was added to make it work.

Take a Look at the Code

The most significant code that was added in this chapter is for deleting a record from the database. In the following sections we will look at that code.

ErrandRsc.h

Open the ErrandRsc.h file in the editor. There is a new `#define` statement for the `MainCompleteButton` – the button that was added to the Main form.

Then there is also a new set of resources for the new form that are listed as the following:

```
#define CompleteForm           <ID>
#define CompleteDoneButton    <ID>
#define CompleteDeleteButton  <ID>
#define CompleteSunCheckbox   <ID>
#define CompleteMonCheckbox   <ID>
#define CompleteTuesCheckbox  <ID>
#define CompleteWedCheckbox   <ID>
#define CompleteThursCheckbox <ID>
#define CompleteFriCheckbox   <ID>
#define CompleteSatCheckbox   <ID>
#define CompleteSunLabel      <ID>
#define CompleteMonLabel      <ID>
#define CompleteTuesLabel     <ID>
#define CompleteWedLabel      <ID>
#define CompleteThursLabel    <ID>
#define CompleteFriLabel      <ID>
#define CompleteSatLabel      <ID>
```

Here is the definition for the new Complete form, its two buttons, the seven checkboxes and the seven labels. One thing that is missing is a group ID resource like we had in Chapter 11. There was a group ID field that was part of the checkbox properties, however, we left this value for these resources as 0. You may recall from Chapter 11 that when a group ID is defined, then only one item in the group can be selected at any one time. Here, we want all of these checkboxes to be checked at the same time, if appropriate. By leaving the group ID for the checkboxes as 0, this is what we were communicating to Constructor. Since no group ID resource was created here, there will be no restriction that only one checkbox be checked.

Finally, there is a new group for the Alert resource – the new “Deleted” alert that we added to the resource file.

Errand.c

Open the Errand.c file in the editor. In the `AppHandleEvent` function, there is a new case for the `CompleteForm`. As with the other forms, we use `FrmSetEventHandler` to set the form handler as `CompleteFormHandleEvent`. This is all the code that was added to this file.

Main.c

Open the Main.c file in the editor. In the `MainFormDoCommand` function, there is a new case for the `MainCompleteButton`. In this case, we use `FrmGotoForm` to switch to the `CompleteForm`. This is the only change to the Main.c code for this chapter.

Complete.c

Open the Complete.c file in the editor. Several functions have been added to control the functionality of this form. Begin with the `CompleteFormHandleEvent` function. First, we get a pointer to the form with `FrmGetActiveForm`. Then there are cases for the `frmOpenEvents` and `ctlSelectEvents` that are sent for this form. With the `frmOpenEvent` we call `FrmDrawForm` to draw the form on the screen. Then, as an initialization procedure, we call `MarkUnusedRows`. This is a function we created to go through the database and check any checkboxes for days that have no information stored

for them. If a `ctlSelectEvent` is sent for the form, those are processed by the `CompleteFormDoButtonCommand` function we've defined.

The `MarkUnusedRows` function takes no parameters and returns nothing, so it is listed as a separate function to keep the code modular, and specifically because we will use it in a couple of places in the project. To start this function, we call `MemPtrNew` to allocate space large enough for the `ErrandDB` struct and get a pointer to that memory location. Then we cycle through the next set of code seven times, one for each of the days of the week. For each day, we call `GetFromDatabase` (our function defined in Chapter 10) to see if there is information in the database. If there's nothing in the database for a specific day, then we get a pointer to the checkbox for the specified day using the `GetObjectPtr` function. Then we call `CtlSetValue` with that pointer and a value of 1. The non-zero value specifies that that box should be checked. After going through each of the days we free the memory that we allocated for the struct pointer using `MemPtrFree`.

The `CompleteFormDoButtonCommand` function is called in response to a `ctlSelectEvent` sent to the `CompleteFormHandleEvent` function. Here there are cases for the `CompleteDoneButton` and the `CompleteDeleteButton`. The `CompleteDoneButton` case calls `FrmGotoFrom` to return to `MainForm`. The `CompleteDeleteButton` case begins by calling our new function `DeleteFromDatabase`, which will delete any records from the database for days that are checked on the form. Then, if there is no error, we call `FrmAlert` to display the `DeletedAlert`. Once this is done, we call `MarkUnusedRows` again to refresh the form with checkboxes selected for days with no entry in the database. For both buttons, everything has been handled, so we return `true` to the event handler.

The `DeleteFromDatabase` is a function that cycles through each day of the week, and calls another function `FindAndDelete` with the day of the week in question and the checkbox that goes with the same day. If there is any error trying to delete any of the checked records, the error code will be returned. Otherwise `errNone` will be returned.

The last new function for this chapter is the `FindAndDelete` function. This function begins by calling `CtlGetValue` on a pointer to the checkbox passed to it. Then the checked variable holds a value that tells us whether the checkbox is checked. If the box is checked, we enter a for loop to go through all the records in the database (the variable `num` was initialized using `DmNumRecords` so we'd know how many records are in the database). In this loop we call `DmQueryRecord` to get a handle to the data in each record. If this handle is valid, we use `MemHandleLock` to get a pointer to the data. Then, if the day that was sent to this function matches a record in the database, there are errands specified for that day. We can delete the record with the handle and pointer we have for it. First we need to call `MemHandleUnlock` to free up the memory. We call `DmRemoveRecord` to get rid of the record completely. In the else case we also call `MemHandleUnlock`, because it is necessary to free up this memory when we are done with it to prevent memory leaks. If there was any error deleting the record, that error code is returned to the calling function.

Errand.h

Open the `Errand.h` file in the editor. Several new function prototypes were added for the `Complete.c` file, including `FindAndDelete`, `DeleteFromDatabase`, `MarkUnusedRows`, `CompleteFormDoButtonCommand`, and `CompleteFormHandleEvent`.

Selector Triggers for Reminders

In this chapter, we will add selector triggers to pop up information from the database to remind the user what errands were set for a particular day. As before, get the sources first to get started.

- ❖ If you followed Chapter 12, go into the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 13\Src folder and copy the Errand.c, Complete.c and Errand.h files to your CodeWarrior Manuals\MW Tutorial for Palm OS\MyErrand\Src folder, overwriting the source files already there.
- ❖ If you skipped Chapter 12, make sure you have a MyErrand folder created in the CodeWarrior Manuals\MW Tutorial for Palm OS folder. Then go to the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 13 folder and copy all the files and folders from that folder into your MyErrand folder.

In this chapter, you will go through the following steps:

- Put Selector Triggers on the Complete Form
- Create a Form to be Triggered by the Selector
- Compile and Run
- Take a Look at the Code

To get started, open the Errand.rcp file in PilRC Designer.

Put Selector Triggers on the Complete Form

Open the Complete form for editing. Place a SELECTORTRIGGER control from the Form category of the Control Palette onto the form and give it the following values:

ID	CompleteSunListSelectorTrigger
Disabled	Unchecked
Text	View Errands...
Usable	Checked
Anchor	caLeft
Height	15
Left	31
Top	21
Width	60

The ellipsis at the end of the label is a common convention used to denote a selector trigger, which is a link to a modal form to display or allow a certain piece of data to be set without cluttering the current form needlessly. We will need several other triggers, so place six more selector triggers onto the form and set their properties according to the

table shown below. The ID field should be set by combining “Complete”+Object Identifier+“SelectorTrigger” as we did for the first selector trigger.

Object Identifier	MonList	TuesList	WedList	ThursList	FriList	SatList
Left Origin	31	31	31	31	31	31
Top Origin	37	54	71	88	105	122
Width	60	60	60	60	60	60
Height	15	15	15	15	15	15
Usable	Checked	Checked	Checked	Checked	Checked	Checked
Anchor	caLeft	caLeft	caLeft	caLeft	caLeft	caLeft
Font	Standard	Standard	Standard	Standard	Standard	Standard
Label	View Errands ...	View Errands...	View Errands ...	View Errands ...	View Errands...	View Errands ...

Once these resources are entered, your form will look like Figure 13.1.

Save the resource file when you are finished.



Figure 13.1: Complete form with selector triggers

Create a Form to be Triggered by the Selector

Create a new form resource and set its ID to “RemindForm”. Give the form the following properties:

Savebehind, Usable, Modal	Checked
Title	Reminder
Height	102

Left	2
Top	56
Width	156

Leave all other fields at their default values. Add a button to the form with these values:

ID	RemindOKButton
Title	OK
Anchor	caLeft
Frame	pfbFrame
Height	12
Left	5
Top	85
Width	36
Font	(Standard)
Usable	Checked

Leave all other fields with the default. Add a label to the form with these values:

ID	RemindDayLabel
Text	Wednesday
Left	2
Top	15
Usable	Checked
Font	(Bold)

We will actually be setting the text of this label dynamically. We need to use the longest string possible for the text that is set here so PilRC Designer does the work of making sure there is enough memory lined up for the label we set during runtime. Since Wednesday is the longest day name, it is the text we put here for that purpose. Add a field to the form, and give it these properties:

ID	RemindErrandField
Field ID	leave default (1503)
Left Origin	2
Top Origin	28
Width	153
Height	52
Usable, Editable	Checked
All others	Unchecked
Max Characters	256
Font	Standard

Leave the other fields as default. The completed form will look like Figure 13.2.

Save your resource file in the Designer before returning to the IDE.



Figure 13.2: The Reminder modal form

Compile and Run

Make the project. Make sure POSE is displaying the App Launcher or the Preferences screen and then run the application on your emulator session. Now, when you click the Complete button, the selector triggers are displayed on the screen. If you click on the trigger for a certain day, you will see the errands listed for that day, or the phrase “No Errands Specified” if nothing has been saved for that day. One example of how this might look is Figure 13.3. Experiment with different days, and notice how the bolded text in the Reminder form shows the day for the trigger you selected (instead of the

“Wednesday” we typed in Constructor). Familiarize yourself with the way the triggers work, and convince yourself that they are displaying the correct data. Then you can proceed to finding out how the code was written to control this behavior.



Figure 13.3: Reminder for a day with errands saved in the database

Take a Look at the Code

The resources in this chapter require some code to make everything happen. That code will be discussed in the following paragraphs.

ErrandRsc.h

Open `ErrandRsc.h` in the editor. In the resource file, there are seven new resources for the seven selector triggers we created. There is also a new group of resources for the modal form we added with its button, label and field.

Errand.c

Open `Errand.c` in the editor. There is a new global variable defined near the beginning of this file. It is called `selectDay`, and is typed as the `Weekday` enumeration we defined. Then, in the `AppHandleEvent` function, there is a new case to define the event handler for the `RemindForm` – `RemindFormHandleEvent`. No other modifications were made to this file.

Complete.c

Open `Complete.c` in the editor. There are several new cases for the `CompleteFormDoButtonCommand` function. One new case was added for each of the seven selector triggers. If one of the triggers is selected, we call `FrmPopupForm` to display the `RemindForm`, while the `CompleteForm` is still displayed in the background. We also set the global variable `selectDay` to the appropriate day for the selected trigger.

`FrmPopupForm` triggers the form handler for this modal form. With a straightforward modal dialog we would have called `FrmDoDialog` instead, and there wouldn't have been any need for an event handler. But we need to do some custom display drawing, so we need the flexibility of our own event handler. `RemindFormHandleEvent` handles the `frmOpenEvent` and the `ctlSelectEvent` for the modal form. In the `frmOpenEvent`, we use `FrmGetFormPtr` to get a pointer to the form (`FrmGetActiveForm` won't work in this case, because `RemindForm` is modal, and there

is no concept of an “active” form with modal forms). Then `FrmDrawForm` is called to display the form. Using `FrmCopyLabel`, we change the text of the label on the form. Instead of displaying “Wednesday” like we set in the label in Constructor, it will display the text from the `dayList` array for the day stored in the `selectDay` array that was set in the `CompleteFormDoButtonCommand` function. Then we call `PutTextForDay` (a function we defined in Chapter 11) to get anything saved in the database for the day in question. That function puts the returned text in the field we specified for it to use, in this case `RemindErrandField`. In the case for a `ctlSelectEvent`, we call `FrmReturnToForm` to close the modal form and restore the `CompleteForm`. We can do this for a blanketed `ctlSelectEvent` because there is no other control on the form that will send this type of event.

Errand.h

Open `Errand.h` in the editor. A new global variable, `selectDay`, has been defined with the `Weekday` type we created. The variable is actually declared in `Errand.c`. Then, a function prototype for `RemindFormHandleEvent` was added.

14

Wrapping Up the Application

In this chapter, we put a few finishing touches on the application. As before, get the sources first to get started.

- ❖ If you followed Chapter 13, there is only one thing you need to do – copy the bitmap files from the `Chapter 14\Rsc` folder to your `MyErrand\Rsc` folder and continue through the chapter.

- ❖ If you skipped Chapter 13, make sure you have a MyErrand folder created in the CodeWarrior Manuals\MW Tutorial for Palm OS folder. Then go to the CodeWarrior Manuals\MW Tutorial for Palm OS\Errand\Chapter 14 folder and copy the MyErrand.mcp file and the Rsc and Src folders from that folder into your MyErrand folder.

In this chapter, you will go through the following steps:

- Create a Form Bitmap
- Create a Large Application Icon
- Create a Small Application Icon
- Compile and Run
- Take a Look at the Code
- Conclusion

To get started, open the Errand.rcp file in PilRC Designer

Create a From Bitmap

Since PilRC Designer does not have its own bitmap editor, we must use an external bitmap editor for all of the bitmaps we create. You may create your own bitmaps, but the images that are shown in this chapter have been created for you. These images should be located in your MyErrand\Rsc folder.

First, create a new Bitmap Family resource. Set the ID of the Bitmap Family to MainCompassBitmapFamily and set the Bimtap1bpp field to Compass-1-bit.bmp and set Bitmapp8bpp to Compass-8-bit.bmp. Both bitmap files should be located in your MyErrand\Rsc folder.

Select the Main form for editing and add a Form Bitmap from the Form category of the Control Palette to the Main Form layout. Give it the following values:

Left Origin	10
Top Origin	36
Bitmap	MainCompassBitmapFamily

When the bitmaps have been pasted in, the layout in Constructor should look like Figure 14.1. Save the resource file when you are finished.



Figure 14.1: Compass bitmap in layout

Create a Large Application Icon

So far we have allowed the default application icon to be used for this application, but as we finalize the application, it would be good to create our own. Instead of Bitmap Family resources, you will need to use Application Icon Family resources. These work very much like bitmap families that we have already worked with, in that you set images of several depths and the OS picks the right one to display, depending on the bit-depth supported by the device.

A new application icon family resource can be created by following these steps:

1. Select the ICONFAMILYEX resource from the Images category of the Control Palette.
2. Set the ID field to "1000" and leave the Compression field as default. (Icon Family resource IDs should always be set to 1000 or 1001.)
3. Next, you will need to insert images into the family. Select "ImageFamilyEx BITMAP" from the Images category of the Control Palette. This will create a new listing in the Design View layout. Since we have not specified a bitmap file yet, you will see the following: "Cannot open file filename.bmp".
4. Select the message "Cannot open file filename.bmp" from the Design View. You should see a box around the text and the Property Editor should change.
5. In the Property Editor for this bmp, make sure the Density field is set to "Density Low", change the BitsPerPixel field to "1" and change the Filename field to the LgAppIcon-1-bit.bmp file that is located in your MyErrand\Rsc folder by clicking the open button (📁) and navigating to it in the window that appears.

6. From the Property Editor dropdown list, choose “LargeAppIconFamily ICONFAMILYEX” so that we can add the high-density version of the bitmap. Notice that the ImageFamilyEx control in the Control Palette is only enabled in the LargeAppIconFamily ICONFAMILYEX view.
7. Repeat steps 3-5, but this time, change the BitsPerPixel field to “8”, leave Density as “Density Low” and change the filename to LgAppIcon-8-bit.bmp (also located in MyErrand\Rsc).

Figures 14.2 and 14.3 show the large B&W and color app icons respectively.

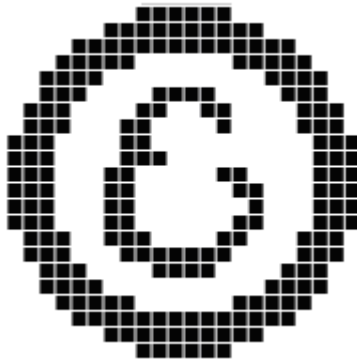


Figure 14.2: The large black and white icon

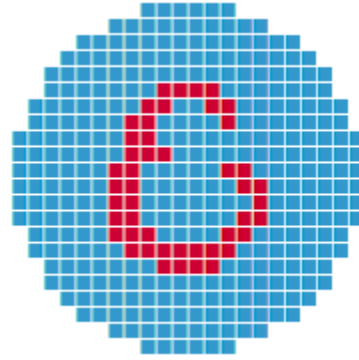


Figure 14.3: The large color icon

Create a Small Application Icon

In Palm OS 3.0 and later, there is a list view for the applications, and this requires a smaller icon than the 22x22 one just created in the previous section. To support this feature, create a second App Icon Family by selecting SMALLICONFAMILYEX. Repeat the same process that was done for the large application icon. This time, set the ID field to “1001”, and the file names will be SmAppIcon-1-bit.bmp and SmAppIcon-8-bit.bmp, both of which can be found in your MyErrand\Rsc folder. Figures 14.4 and 14.5 show the small B&W and color app icons respectively.

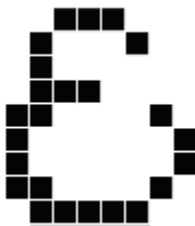


Figure 14.4: Small B/W icon



Figure 14.5: Small color icon

Save and close the resource file before returning to the IDE.

Compile and Run

Make the project. Make sure POSE is displaying the App Launcher or the Preferences screen and then run the application on your emulator session. If you run the application on POSE with a color ROM file, your screen should look like Figure 14.7.



Figure 14.7: Running app on color device

You will also want to return to the App Launcher screen to see the icons we created for this program. To see the small icon, you will need to switch to the list view. On a device running OS 3.0 or later, do this by pulling up the menu on the App Launcher screen. Choose Options-Preferences, and change the “View By” popup trigger to “List”.

Take a Look at the Code

There is no code that was changed to display the various bitmaps that were set up in this chapter. However, a few resources were added, so we will look at the modifications to the header file Constructor created.

ErrandRsc.h

Open ErrandRsc.h in the editor. In the new set of resources, there is a new line for the MainComapass1Bitmap and MainCompass2Bitmap resources, which are the form bitmaps we added for the compass picture. Further down the file, there are several new resources for the new bitmaps that were created as family elements for the bitmap family. The new resource groups are for the bitmap family that was created for the compass form bitmap.

Conclusion

We've come a long way since the blank screen in Chapter 2. Hopefully, you've learned a few things about the way resources and code come together to create a Palm OS application. Now you're ready to start into your own development. Happy coding!