

CodeWarrior® Development Studio Targeting the Palm OS® Platform

Revised 20021206



Metrowerks, the Metrowerks logo, and CodeWarrior are registered trademarks of Metrowerks Corp. in the US and/or other countries. Graffiti, HotSync, Palm OS, Palm Modem, 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 tradenames and trademarks are the property of their respective owners.

Copyright © Metrowerks Corporation. 2002. ALL RIGHTS RESERVED.

The reproduction and use of this document and related materials are governed by a license agreement media, it may be printed for non-commercial personal use only, in accordance with the license agreement related to the product associated with the documentation. Consult that license agreement before use or reproduction of any portion of this document. If you do not have a copy of the license agreement, contact your Metrowerks representative or call 800-377-5416 (if outside the US call +1 512-997-4700). Subject to the foregoing non-commercial personal use, no portion of this documentation may be reproduced or transmitted in any form or by any means, electronic or mechanical, without prior written permission from Metrowerks.

Metrowerks reserves the right to make changes to any product described or referred to in this document without further notice. Metrowerks makes no warranty, representation or guarantee regarding the merchantability or fitness of its products for any particular purpose, nor does Metrowerks assume any liability arising out of the application or use of any product described herein and specifically disclaims any and all liability. **Metrowerks software is not authorized for and has not been designed, tested, manufactured, or intended for use in developing applications where the failure, malfunction, or any inaccuracy of the application carries a risk of death, serious bodily injury, or damage to tangible property, including, but not limited to, use in factory control systems, medical devices or facilities, nuclear facilities, aircraft or automobile navigation or communication, emergency systems, or other applications with a similar degree of potential hazard.**

USE OF ALL SOFTWARE, DOCUMENTATION AND RELATED MATERIALS ARE SUBJECT TO THE METROWERKS END USER LICENSE AGREEMENT FOR SUCH PRODUCT.

How to Contact Metrowerks

Corporate Headquarters	Metrowerks Corporation 9801 Metric Blvd. Austin, TX 78758 U.S.A.
World Wide Web	http://www.metrowerks.com
Ordering & Technical Support	Voice: (800) 377-5416 Fax: (512) 997-4901

Table of Contents

1	Introduction	9
	Overview of This Manual	10
	Related Documentation.	11
	Documentation Organization	11
	Where to Go from Here	12
	CodeWarrior IDE Overview	13
	The CodeWarrior Build Process	14
	The Old Build Tools	14
	The New Build Tools	17
	Development Tools for Palm OS®.	19
	CodeWarrior IDE	20
	CodeWarrior Command-Line Tools	20
	Macintosh 68K Compiler	21
	PilRC Compiler	22
	PilRC Designer	22
	Rez Compiler	23
	Palm OS 68K Linker	23
	Macintosh 68K Linker	23
	Palm OS ARM Compiler and Linker	24
	Mac OS Merge Linker	25
	PalmRez Post-Linker	25
	Palm OS Emulator	26
	Palm OS Simulator	26
	Constructor for Palm OS	27
	Object Library for Palm OS	27
	Conduit Development Kit	28
2	Working With Projects	29
	Types of Palm OS® Projects	30
	Single-Segment Applications	30
	Multi-Segment Applications	30

Code Resources	31
Static Libraries	31
Shared Libraries	32
Hacks	32
Working With Palm OS Stationery	34
Stationery Reference	34
Using Stationery	36
Working With Palm OS Wizards	38
Converting Older Projects to Version 9	40
3 Working With Applications	43
Types of Application Projects	44
Single-Segment Applications	44
Multi-Segment Applications	44
Creating Application Projects	45
Object Library for Palm OS Application Wizard	46
Palm OS C and C++ Application Wizards	51
Single-Segment Application Limitations	55
Changing the link order of the application	55
Using the Smart code model	56
Reorganizing or rewriting the application	56
Using Jump Islands	56
Converting Single-Segment Applications to Multi-Segment Applications	57
Adding ARMlets to Existing Projects	58
Open the Project	59
Create a New Build Target	60
Configure the Build Target Linker	62
Configure the Build Target Access Paths	66
Add the ARMlet Runtime Library	72
Write ARMlet Source Code	76
Add ARMlet Source Code to the Project	78
Create Build Target Dependencies	79
Merge the ARMlet Executable Code Into the Application	81

Build the Application Target	87
4 Working With Libraries	89
Types of Libraries	90
Static Libraries	90
Shared Libraries	91
Creating Static Library Projects	92
Creating Shared Library Projects	94
Runtime Libraries	98
Runtime Library Reference	98
Multi-Segment Project Libraries	99
MSL Libraries	101
Standard C Library	101
Standard C++ Library	103
5 Working With Resources	105
Resource Guidelines	106
Creating Palm OS® Resources	107
PilRC Designer	107
Constructor for Palm OS	107
Other Utilities	108
Adding Resource Files To Projects	109
Converting Resource Files	110
History of Palm OS Resource Compilers	110
Using Conversion Utilities	111
Using the Mac OS Merge Linker and PalmRez Post-Linker	113
6 Target Settings Reference	119
Other Target Settings Documentation	120
Target Settings Window Overview	121
Target Settings	123
Target Name	124
Linker	124
Pre-Linker	125

Table of Contents

Post-Linker	125
Save Project Entries Using Relative Paths	126
68K Target.	127
Palm OS Application	128
Palm OS Code Resource	130
Palm OS Static Library	132
Mac OS Application	134
Mac OS Code Resource	136
Palm OS 68K Target.	139
68K Application (Standard, Expanded, or Exp. A5-Jumptable)	140
68K Shared Library	142
68K Code Resource	144
Merged Resource File	146
68K Static Library (Standard, Expanded, or Exp. A5-Jumptable)	148
Palm OS ARMlet Target	150
Output File Name	151
Generate Link Map	151
PRC File Settings	153
Creator	154
PilRC Settings	155
Rez	157
68K Processor	159
Code Model	161
PC-Relative Strings	162
PC-Relative Constant Data	163
68K Disassembler.	164
68K Linker	166
Generate Link Map	167
Dead-Strip Static Initialization Code	168
ELF Disassembler.	169
PalmRez Post-Linker	171
Palm OS Debugging	174

7 C and C++ Compilers	177
Language Extensions	179
Inline Data	179
Specifying the Registers for Arguments	180
Inline Assembly	181
Inline Assembly Instructions	181
Inline Assembly Syntax	182
Inline Assembly Directives	187
Number Formats	190
Integer Formats	190
68K Floating Point Formats	192
Calling Conventions	193
Variable Allocation	194
Register Variables	194
Floating Point Support	195
Pragma Directives	196
warn_a5_access	196
warn_stack_usage	196
C++ Issues for Palm OS	197
Expanded Global Space Mode	198
Reducing Global Space Usage	198
Callback Thunks	198
8 Debugging	201
Debugger Limitations	202
Debugging Palm OS® Projects	203
Debugging Using the Palm OS Emulator	203
Debugging Using the Palm OS Simulator	214
Debugging Using a Palm OS Device	221
Stepping into Palm OS Traps	227
Debugging Shared Libraries and Code Resources	229
Building the Host Application	229
Building the Shared Library or Code Resource	230

Defining the Host Application	230
Starting the Debug Session	231
Using the Palm OS® Debug Console	232
A Troubleshooting	235
Looking Up Palm OS Error Codes	236
Debugger Problems	237
Stop Command Does Not Work	237
Cannot Access Serial Port Error Message	238
Problems After Cancelling a Transfer	238
Short Battery Life	239
Device Display Flashes After Quitting or Resetting	240
Compiling and Linking Problems.	241
Projects From Earlier Releases Do Not Link	241
Linker Error: 16-bit Reference is Out of Range	242
Linker Error: __RuntimeModule__: Entry Point '__InitCode__' is undefined	243
Link Errors After Deleting a .tmp File (Windows only)	244
Postlinker Error: Error Copying resource.frk	244
Other Problems	245
Application Crashes When Launched	245
Index	247

Introduction

This manual describes how to use the CodeWarrior IDE to build different types of Palm OS® executable files. These are the sections in this introduction chapter:

- [Overview of This Manual](#)—describes the contents of this manual
- [Related Documentation](#)— describes supplementary CodeWarrior documentation, third-party documentation, and references to helpful code examples and web sites
- [CodeWarrior IDE Overview](#)— describes the CodeWarrior compiler architecture
- [The CodeWarrior Build Process](#) — describes how the CodeWarrior IDE builds Palm OS executables
- [Development Tools for Palm OS®](#) — provides an overview of the CodeWarrior development tools for the Palm OS, including descriptions of each of the major components in the CodeWarrior IDE

Overview of This Manual

This manual contains information specific to Palm OS software development. [Table 1.1](#) describes the information provided by each chapter in this manual.

Table 1.1 Contents of chapters in this manual

Chapter	Description
Introduction	This chapter provides an introduction to the current version of the CodeWarrior Development Tools for Palm OS tools, describes how the tools work, and provides references to other helpful documentation.
Working With Projects	This chapter provides an overview of the Palm OS project stationery and wizards, shows how to create projects, and shows how to perform common project configuration tasks.
Working With Applications	This chapter describes how to create and configure Palm OS application projects.
Working With Libraries	This chapter describes how to use various runtime libraries in your CodeWarrior projects.
Working With Resources	This chapter describes how to create resources for Palm OS and how to use resource files in CodeWarrior projects.
Target Settings Reference	This chapter describes the target settings panels specific to developing software for the Palm OS.
C and C++ Compilers	This chapter describes the C and C++ back-end compilers and linkers for Palm OS software development.
Debugging	This chapter describes how to use the CodeWarrior IDE to debug Palm OS code.
Troubleshooting	This appendix provides troubleshooting information specific to developing Palm OS software with the CodeWarrior IDE.

Related Documentation

This section provides descriptions of supplementary CodeWarrior documentation and third-party documentation as well as references to example source code and helpful web sites. This section consists of the following topics:

- [Documentation Organization](#)
- [Where to Go from Here](#)

Documentation Organization

The CodeWarrior IDE is a multi-host, multi-language, multi-target development environment. For a complete understanding of the CodeWarrior environment, you must refer to both the general documentation and the documentation that is specific to the target for which you wish to write code (such as this manual).

This manual is organized so that various chapters in this manual are extensions of particular generic manuals, as shown in [Table 1.2](#). For a complete discussion of a particular subject, you may need to read both the generic manual and the corresponding chapter in this manual.

Table 1.2 Related CodeWarrior documentation

Chapter	Related Manuals
Introduction Working With Projects Working With Applications Working With Libraries Working With Resources Debugging	<i>IDE User's Guide</i>
Target Settings Reference	<i>IDE User's Guide</i> <i>C Compilers Reference</i>
C and C++ Compilers	<i>C Compilers Reference</i>

Where to Go from Here

All the manuals mentioned below are also located on the *CodeWarrior Reference CD*.

- Due to significant changes in the build tools in this version of CodeWarrior Development Tools for Palm OS, all users should read the rest of this chapter carefully before moving on to other sections of the manual. In particular, you should read [“The CodeWarrior Build Process” on page 14](#) and [“Development Tools for Palm OS®” on page 19](#).
- All users should read the product release notes, located here (where *CWFolder* is the folder where you installed the CodeWarrior Development Tools for Palm OS package):
`CWFolder\Release Notes\`
- For general information about the CodeWarrior IDE, see the *IDE User's Guide*.

If you are new to the CodeWarrior IDE:

- For details on how to use the CodeWarrior IDE, refer to the *IDE User's Guide*.
- For an overview of how the CodeWarrior development environment for Palm OS works, see [“Development Tools for Palm OS®” on page 19](#).
- CodeWarrior University offers free, online supplemental courses in a variety of programming disciplines:
<http://www.codewarrioru.com/>

If you are new to Palm OS Programming:

- Take the *Palm OS Tutorial*. You can find it here:
`CWFolder\CodeWarrior Manuals\MW Tutorial for Palm OS\CW_Palm_OS_Tutorial.pdf`
- For 68000 assembly language instructions, read the *M68000 Family Programmer's Reference Manual*, located here:
`CWFolder\CW for Palm OS Support\Documentation\Motorola CPU Docs\M68000PRM.pdf`
- Here are some web resources you may find useful:
<http://oasis.palm.com/dev/kb>
<http://dev.palmos.com>
<http://groups.yahoo.com/group/palm-dev-forum/>

CodeWarrior IDE Overview

The CodeWarrior IDE is the integrated development environment application that allows you to develop software. The IDE is comprised of a project manager, source code editor, class browser, various compiler and linkers, and a source-level debugger. The IDE has an extensible architecture that uses plug-in compiler and linker modules to target various operating systems and platforms. The plug-in compiler and linker modules included with CodeWarrior Development Tools for Palm OS allow you to compile code for Palm OS.

The project manager may be new to those more familiar with command-line development tools. It organizes all project-related files and settings. The Project window allows you to see the project at a glance, and eases the organization of and navigation among source code files. The project manager manages all build dependencies automatically.

CodeWarrior projects may contain multiple build targets. A *build target* is a separate build (with its own settings) that uses some or all of the files in the project. For example, CodeWarrior developers commonly implement a debug version and a release version of a piece of software as separate build targets in the same project.

A proprietary, multi-language, multi-target compiler architecture is at the heart of the CodeWarrior IDE. Front-end language compilers generate a memory-resident, unambiguous, language-independent intermediate representation (IR) of syntactically-correct source code. Back-end compilers generate object code from the IR for specific targets. As a result of this architecture, the same front-end compiler may support multiple back-end compilers. In some cases, the same back-end compiler can generate code from a variety of front-end compilers and languages. CodeWarrior linkers generate final executable files from the object code. The CodeWarrior IDE manages the whole process. Multiple linkers that support different object code formats are available for some targets.

All CodeWarrior compilers and linkers are built as plug-in modules. The interface between the IDE and compilers and linkers is public; so third parties can create compilers that work with the CodeWarrior IDE. To learn how to write your own CodeWarrior plug-ins, refer to the *Extending the CodeWarrior IDE* manual. The CodeWarrior Software Development Kit (SDK) is included on the CodeWarrior installer CD, and is also available on the Metrowerks web site at:

<http://www.metrowerks.com/MW/Support/API.htm>

For information about general CodeWarrior IDE features, refer to the *IDE User's Guide*.

The CodeWarrior Build Process

This section describes the process the CodeWarrior IDE uses to build Palm OS executables.

The build tools have changed significantly in this version of CodeWarrior Development Tools for Palm OS. This section describes both the old and new build processes to show how the new process differs from the old (CodeWarrior Development Tools for Palm OS version 8 and lower) build process.

While CodeWarrior Development Tools for Palm OS has support for the old process, we recommend that you use the new process for all new Palm OS projects.

The Old Build Tools

[Figure 1.1 on page 16](#) shows an illustration of the old CodeWarrior Development Tools for Palm OS build process and highlights the tools that CodeWarrior Development Tools for Palm OS version 8 and lower uses to build Palm OS executables. The old build process centers around the [Macintosh 68K Linker](#).

While this version of CodeWarrior Development Tools for Palm OS still supports the old build tools and build process, you should use the new build tools to create all new projects. Also, we recommend that you convert existing projects to use the new tools. This involves selecting **Palm OS 68K Linker** from the [Linker](#) menu in the [Target Settings](#) panel, and setting up the [Palm OS 68K Target](#) panel in all build targets in a project. You may need to set up other target settings panels, depending on the type of project. See [“Converting Older Projects to Version 9” on page 40](#) for more information on converting older projects to this version.

Compile Stage

During the compile stage of the old build process, the [Macintosh 68K Compiler](#) translates C and C++ source code from the project into object code.

The [PilRC Compiler](#) translates resource compiler for Palm OS source (RCP) files into Rez source code files.

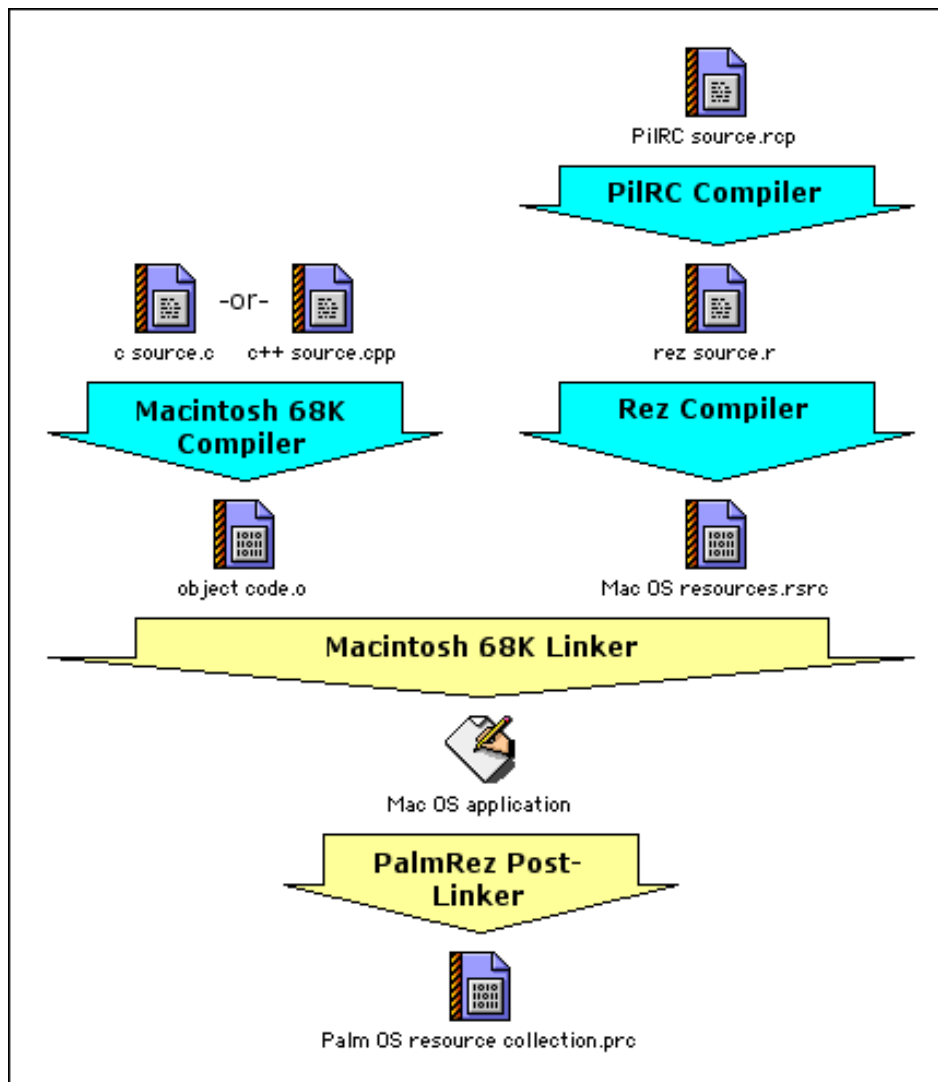
The [Rez Compiler](#) then compiles the Rez source code files into the Macintosh resource files. You can edit the resulting Macintosh resource files using [Constructor for Palm OS](#).

Link Stage

During the link stage of the build, the [Macintosh 68K Linker](#) first links the object files and static libraries in the project to produce Macintosh ‘code’ and ‘data’ resources. The linker then merges the Macintosh resource files from the [Rez Compiler](#) and any Macintosh resource files in the project with the Macintosh ‘code’ and ‘data’ resources to create a Macintosh 68K application.

Next, the [PalmRez Post-Linker](#) translates the Macintosh 68K application resources into a Palm OS resource collection (PRC) file suitable for running on a Palm OS device, simulator, or emulator.

Figure 1.1 Old CodeWarrior Development Tools for Palm OS build process illustrated



The New Build Tools

[Figure 1.2 on page 18](#) shows an illustration of the new build process and highlights the tools that this version of CodeWarrior Development Tools for Palm OS uses to build Palm OS executables. The new build process centers around the new [Palm OS 68K Linker](#).

You should use the new build tools outlined in this section and in [“Development Tools for Palm OS®” on page 19](#) to create all new projects.

Compile Stage

During the compile stage of the build, the [Macintosh 68K Compiler](#) translates C and C++ source code from the project into object code.

The [PilRC Compiler](#) behaves differently in the new build process, in that it translates resource compiler source (RCP) files into intermediate resource object (RO) files instead of Rez source files. These intermediate resource object files are essentially Palm OS resource collection (PRC) files without any ‘code’ or ‘data’ resources in them.

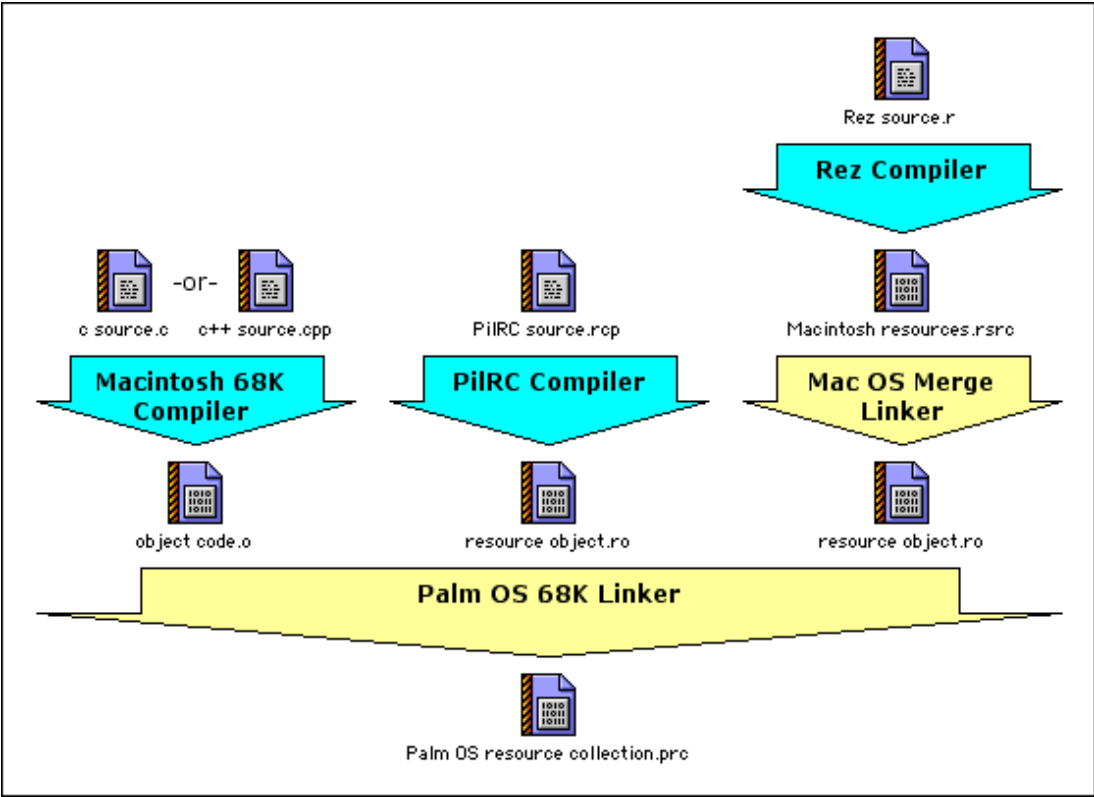
The [Rez Compiler](#) behaves the same as it does in the old build process — it translates any Rez source code files in the project into Macintosh resource files.

Link Stage

During the link stage of the build, the [Palm OS 68K Linker](#), a new linker based on the [Macintosh 68K Linker](#) in the old tool set, first links the object files and static libraries in the project to produce Macintosh ‘code’ and ‘data’ resources. The linker then merges the resources from the RO files that the [PilRC Compiler](#) output in the compile stage of the build process with the ‘code’ and ‘data’ resources that the [Palm OS 68K Linker](#) output to create a final PRC file.

Unlike the [Macintosh 68K Linker](#), the [Palm OS 68K Linker](#) does not automatically merge Macintosh resource files in the project into the final PRC file. You can still use the [Rez Compiler](#) to convert Rez source files into Macintosh resource (.rsrc) files; however the [Palm OS 68K Linker](#) does not merge them into the final PRC file. For instructions on how to convert Macintosh resources to resource compiler source (RCP) files that the PilRC compiler merges into the final PRC file, see [“Converting Resource Files” on page 110](#).

Figure 1.2 New CodeWarrior Development Tools for Palm OS build process illustrated



Development Tools for Palm OS®

The CodeWarrior Development Tools for Palm OS package contains many tools you can use to develop Palm OS® software.

NOTE	The development tools for Palm OS have changed significantly in this version of CodeWarrior Development Tools for Palm OS. While we support the old tools for older projects, we prefer that you use the new tools for all new development. Please read this section carefully to learn about the new tools and how they work.
-------------	--

This section provides an overview of these tools:

- [CodeWarrior IDE](#)
- [CodeWarrior Command-Line Tools](#)
- [Macintosh 68K Compiler](#)
- [PiLRC Compiler](#)
- [PiLRC Designer](#)
- [Rez Compiler](#)
- [Palm OS 68K Linker](#)
- [Macintosh 68K Linker](#)
- [Palm OS ARM Compiler and Linker](#)
- [Mac OS Merge Linker](#)
- [PalmRez Post-Linker](#)
- [Palm OS Emulator](#)
- [Palm OS Simulator](#)
- [Constructor for Palm OS](#)
- [Object Library for Palm OS](#)
- [Conduit Development Kit](#)

CodeWarrior IDE

The CodeWarrior IDE is an application that allows you to write and build software. It contains a project manager, a source code editor, a class browser, various compilers and linkers, and an integrated debugger.

The concept of a project manager may be new to those more familiar with command-line development tools. The project manager organizes all files and settings related to a project. It allows you to view your project at a glance, and eases the organization of and navigation among your source code files. The CodeWarrior project manager manages all build dependencies automatically.

A CodeWarrior project may have multiple build targets. A *build target* is a separate build with its own settings that uses some or all of the files in the project. For example, you might create separate build targets for a debug version and a release version of your software.

The CodeWarrior integrated debugger controls program execution and allows you to view internal data within your program as it runs. You use the debugger to find problems in your program by executing your program one statement at a time, or by suspending execution at specified breakpoints. When you suspend a program, the debugger lets you view the stack, change variables and memory contents, and inspect the contents of the processor registers.

CodeWarrior Command-Line Tools

The CodeWarrior Development Tools for Palm OS command-line compilers and linkers are command-line versions of the CodeWarrior compiler and linker plug-in modules. We provide these tools for developers who prefer to use make files rather than CodeWarrior IDE projects to generate software for the Palm OS platform.

The command-line compilers translate source code into object code, storing the object code in files. The command-line linkers combine one or more object code files to produce an executable file such as an application, shared library, code resource, or static library.

NOTE	The command-line tools obtain settings information from command-line switches rather than from the CodeWarrior IDE settings panels.
-------------	---

The *C Compilers Reference* manual contains information about how to access and use the CodeWarrior command-line tools.

Macintosh 68K Compiler

The Macintosh 68K compiler is a CodeWarrior ANSI-compliant C/C++ compiler plug-in module that is integrated into the CodeWarrior IDE. This compiler is based on the same compiler architecture that is used in all of the CodeWarrior C/C++ compilers. The compiler supports statement-level and function-level inline assembly language as described in [“Inline Assembly” on page 181](#).

In the old build process (see [“The Old Build Tools” on page 14](#)), the IDE uses this compiler with the [Macintosh 68K Linker](#) to produce Mac OS 68K application files that the [PalmRez Post-Linker](#) transforms into Palm OS executable resource collection (PRC) files.

NOTE	You should only use the Macintosh 68K Linker for existing Palm OS projects.
-------------	---

In the new build process (see [“The New Build Tools” on page 17](#)), the IDE uses this compiler to translate C and C++ source code into object code for the Motorola 68K family of processors used in Palm OS devices. The [Palm OS 68K Linker](#) then translates the object code into Palm OS executable resource collection (PRC) files.

NOTE	The Palm OS 68K Linker is the preferred linker for Palm OS development.
-------------	---

You enable this compiler when you select either **Macintosh 68K** or **Palm OS 68K** from the **Linker** menu in the [Target Settings](#) panel. See [“68K Processor” on page 159](#) and [“68K Disassembler” on page 164](#) for information about how to configure this compiler. For more information on this compiler, see [“C and C++ Compilers” on page 177](#).

PilRC Compiler

PilRC is an acronym for *Pilot Resource Compiler*. The PilRC compiler is a CodeWarrior plug-in module that compiles Palm OS resources.

In the old build process (see [“The Old Build Tools” on page 14](#)), the PilRC compiler compiles resource compiler source (RCP) files into Rez source (`.r`) files that the [Rez Compiler](#) later turns into Macintosh resource (`.rsrc`) files.

In the new build process (see [“The New Build Tools” on page 17](#)), the PilRC compiler compiles RCP files into intermediate resource object (RO) files that the [Palm OS 68K Linker](#) later adds to the final Palm OS resource collection (PRC) file.

You enable this compiler when you select **Macintosh 68K** or **Palm OS 68K** in the **Linker** menu in the [Target Settings](#) panel. Refer to [“PilRC Settings” on page 155](#) for information on configuring this compiler.

PilRC Designer

PilRC Designer is a visual resource editing tool that is installed with the CodeWarrior Development Tools for Palm OS package. You use PilRC Designer to create the user interfaces for your Palm OS projects. It provides a WYSIWYG (What You See Is What You Get) view of a Palm Pilot organizer’s screen onto which you lay out forms, alerts, and other interface objects such as bitmaps.

This is the preferred resource editor for projects that use the new build tools (see [“The New Build Tools” on page 17](#)) because it outputs resources in RCP files that you can include in your CodeWarrior projects.

To run PilRC Designer, select **PalmOS > Launch PilRC Designer** from the CodeWarrior menu bar.

PilRC Designer is located in the following folder (where *CWFolder* is the folder where you installed the CodeWarrior Development Tools for Palm OS package):

```
CWFolder\CW For Palm OS Tools\PilRC Designer\
```

Rez Compiler

The Rez compiler is a CodeWarrior plug-in module that compiles Rez source (`.r`) files into Macintosh resource (`.rsrc`) files.

You enable this compiler when you select **Macintosh 68K** in the **Linker** menu in the [Target Settings](#) panel. Refer to [“Rez” on page 157](#) for information on configuring this compiler.

Palm OS 68K Linker

The Palm OS 68K linker is a new CodeWarrior plug-in module that is integrated into the CodeWarrior IDE. The linker links multiple object code modules, libraries, and binary resource object (RO) files, resolving references from one to another to produce executable applications, shared libraries, static libraries, or code resources.

This linker is the preferred linker for creating Palm OS executables. This linker does not generate intermediate Mac OS executables during a build and therefore does not require the [PalmRez Post-Linker](#). Instead, the linker works with Palm OS formats throughout a more stream-lined build process, supporting Macintosh-format resources only for compatibility with older projects.

You enable this linker when you select **Palm OS 68K** in the **Linker** menu in the [Target Settings](#) panel. To learn how to configure this linker, see [“68K Linker” on page 166](#). For information on the [PalmRez Post-Linker](#) settings panel, see [“PalmRez Post-Linker” on page 171](#).

Macintosh 68K Linker

The Macintosh 68K linker is a plug-in module that is integrated into the CodeWarrior IDE. This linker links multiple object code modules, libraries, and Macintosh resource (`.rsrc`) files, resolving references from one to another to produce executable applications, shared libraries, static libraries, or code resources. The Macintosh 68K linker uses the [PalmRez Post-Linker](#) to generate Palm OS resources from the resources in the project.

NOTE	Use this linker only for existing Palm OS projects.
-------------	---

You enable this linker when you select **Macintosh 68K** in the **Linker** menu in the [Target Settings](#) panel. To learn how to configure the this linker, see [“68K Linker” on page 166](#). For information on the [PalmRez Post-Linker](#) settings panel, see [“PalmRez Post-Linker” on page 171](#).

Palm OS ARM Compiler and Linker

The Palm OS ARM linker and compiler are new CodeWarrior plug-in modules that are integrated into the CodeWarrior IDE. The compiler translates C and C++ source code into object code for the ARM processors used in Palm OS devices. The linker links multiple ARM object code modules and libraries, resolving references from one to another to produce executable applications, shared libraries, static libraries, or code resources.

Next-generation Palm OS devices are based on Advanced RISC Machines (ARM) processors replacing the Motorola Dragonball 68K processor, which has powered all Palm devices in the past.

Although the underlying processor is different in the next-generation Palm OS devices, very few things have changed for application developers developing for Palm OS 5. Palm OS 5 includes the Palm Application Compatibility Environment (PACE). The PACE is an interpretation layer that offers the following interactions between 68K code, ARM-native code, and the Palm OS 5 operating system, allowing the majority of Palm OS 68K code to run unchanged on ARM-based devices:

- allows 68K application code to run on Palm OS 5, an operating system that is ARM-native code
- allows 68K code to call ARM routines in the form of ARM-native code resources (ARMlets)
- allows ARMlets to call Palm OS 5 system routines

Developers who want to write specialized code that is optimized for ARM processors (an optimized Fast Fourier Transform function, file compression code, or graphics algorithms for instance, where there are very few Palm OS 5 system calls) can add ARMlets to existing 68K applications to gain the benefit of fast-running, ARM-native code.

You enable the linker when you select **Palm OS ARMlet** in the **Linker** menu in the [Target Settings](#) panel.

To learn how to configure this linker, see [“Palm OS ARMlet Target” on page 150](#).

For instructions on how to create an ARMlet and add it to an existing CodeWarrior project, see [“Adding ARMlets to Existing Projects” on page 58](#).

Mac OS Merge Linker

The Mac OS Merge linker is a CodeWarrior linker plug-in module that is integrated into the CodeWarrior IDE. The linker merges Macintosh resources from resource (.rsrc) files in a build target and places them into the final output file.

In the old build process (see [“The Old Build Tools” on page 14](#)), you only need to use the Mac OS Merge linker if you want to generate a stand-alone Macintosh resource (.rsrc) file directly from the output of the [Rez Compiler](#) for use with [Constructor for Palm OS](#). The [Macintosh 68K Linker](#) automatically processes the output of the [Rez Compiler](#), any Macintosh resource files in the project.

In the new build process (see [“The New Build Tools” on page 17](#)), the Mac OS Merge linker and PalmRez post-linker work together to translate Macintosh resources into intermediate resource object (RO) files that the [Palm OS 68K Linker](#) later adds to the final Palm OS resource collection (PRC) file.

You enable this linker when you select **Mac OS Merge** in the **Linker** menu in the [Target Settings](#) panel. Refer to [“Target Settings” on page 123](#) for more information.

PalmRez Post-Linker

The PalmRez post-linker is a CodeWarrior plug-in module that is integrated into the CodeWarrior IDE. The post-linker runs at the end of the link stage of a build. This post-linker converts Macintosh applications that the [Macintosh 68K Linker](#) generates to the final Palm OS resource collection (PRC) file.

NOTE	Use this post-linker only for existing Palm OS projects. This post-linker only works with the Macintosh 68K Linker , which is part of the old build tool set. For all new Palm OS projects, use the Palm OS 68K Linker .
-------------	--

Palm OS Emulator

The *Palm OS Emulator* (POSE) allows your computer to emulate a Palm OS handheld device. POSE lets you test an application as if it were running on a handheld, but without inconveniences such as manual resets and long download times. With it, you can test applications across the entire range of Palm OS versions up to, but not including, Palm OS 5. When you run an application in the Palm OS Emulator, the emulator attempts to detect things in the application that may cause problems on Palm OS devices.

When the current build target is configured to use the Palm OS Emulator, you can run it by selecting **PalmOS > Launch Emulator** from the CodeWarrior menu bar. See [“Debugging Using the Palm OS Emulator” on page 203](#) for instructions on how to use POSE to debug your CodeWarrior projects.

Palm, Inc. regularly updates the Palm OS Emulator software. For the latest version, please go to this URL:

<http://www.palmos.com/dev/tech/tools/emulator>

Palm OS Simulator

The *Palm OS Simulator* is a Windows application that contains the Palm OS 5.0 operating system compiled natively for Intel hardware. The simulator is not a hardware emulator like the [Palm OS Emulator](#); but rather the real operating system running natively on top of a data abstraction layer on Intel hardware.

We recommend that you use both [Palm OS Emulator](#) and Palm OS Simulator to debug and test your applications, because, unlike the Palm OS Emulator, the Palm OS Simulator does not do nearly as much checking and validating of applications. While Palm OS Emulator does its best job to find things in applications that may cause problems on Palm OS devices, there are some problems that are very difficult to detect. The Palm OS Simulator helps you detect such problems.

When the current build target is configured to use the Palm OS Simulator, you can run it by selecting **PalmOS > Launch Emulator** from the CodeWarrior menu bar. See [“Debugging Using the Palm OS Simulator” on page 214](#) for instructions on how to debug your CodeWarrior projects using the Palm OS Simulator.

For more information about the Palm OS Simulator, go to this URL:

<http://www.palmos.com/dev/tools/simulator/>

Constructor for Palm OS

Constructor for Palm OS is a graphical design tool that lets you build the visual elements of a Palm OS application such as forms, help text, alerts, icons, and bitmap pictures.

Select **PalmOS > Launch Constructor for Palm OS** from the CodeWarrior menu bar to run Constructor for Palm OS. The application is located here (where *CWFolder* is the folder where you installed the CodeWarrior Development Tools for Palm OS package):

`CWFolder\CW for Palm OS Tools\Constructor for Palm OS\`

The *Constructor for Palm OS* manual is available in PDF form here:

`CWFolder\CW for Palm OS Support\Documentation\Palm OS 5.0
Docs\Constructor for Palm OS.pdf`

Object Library for Palm OS

Object Library for Palm OS (POL) is a C++ class library for Palm OS development that is bundled with the CodeWarrior IDE. POL speeds up development by reducing the effort required to create and maintain Palm OS programs. Another benefit of POL is that complex applications require less debugging.

For example, a developer who writes the typical "Hello World" application for Palm OS in traditional style must write over 160 lines of code. A developer who uses the POL to write the same application may do it in a mere 16 lines of code!

The POL is organized like the most of known C++ libraries for Windows platform, such as MFC, WTL, and others. POL is a framework that supports the entire application, implementing a fundamental, strategical, new approach for writing Palm OS applications. This approach allows developers to write applications in a strict object-oriented style.

Select **Help > Object Library for Palm OS Reference** from the CodeWarrior menu bar to view an online version of the Object Library for Palm OS reference guide.

The Object Library for Palm OS Tutorial is included in the following location (where *CWFolder* is the folder where you installed the CodeWarrior Development Tools for Palm OS package):

`CWFolder\CodeWarrior Manuals\Object Library for Palm OS
Tutorial\`

For more information about POL, see this web site:

<http://www.agpoint.com/pol/>

Conduit Development Kit

Conduits exchange and synchronize the data between a desktop application and a Palm OS handheld. Conduits are developed as plug-ins for the HotSync Manager desktop application.

To develop conduit plug-ins, you may use CodeWarrior for Windows or CodeWarrior for Mac OS. This version of CodeWarrior Development Tools for Palm OS includes an add-on kit that you can apply to CodeWarrior for Windows to develop conduit applications.

More information about conduits can be found in the Conduit Development Kit (CDK) which you can download here:

<http://www.palmos.com/dev/tech/conduits/>

Working With Projects

This chapter describes various issues related to working with Palm OS® projects with the CodeWarrior IDE. The sections in this chapter are:

- [Types of Palm OS® Projects](#) — describes the different types of Palm OS projects you can create using the CodeWarrior IDE, and provides references to other sections in this manual that relate to those types of projects
- [Working With Palm OS Stationery](#) — describes the CodeWarrior stationery projects available for Palm OS development and shows how to use them
- [Working With Palm OS Wizards](#) — describes the various CodeWarrior wizards included in the CodeWarrior Development Tools for Palm OS package and shows how to use them
- [Converting Older Projects to Version 9](#) — shows how to update older Palm OS projects to this version of CodeWarrior Development Tools for Palm OS

Types of Palm OS® Projects

This section describes the different types of CodeWarrior projects you can create with the CodeWarrior Development Tools for Palm OS package.

You can create the following types of projects:

- [Single-Segment Applications](#)
- [Multi-Segment Applications](#)
- [Code Resources](#)
- [Static Libraries](#)
- [Shared Libraries](#)
- [Hacks](#)

Single-Segment Applications

A *single-segment application* is a Palm OS application that contains a single code segment that is smaller than 64 kilobytes in size. This is the most common type of application on the Palm OS.

To learn how to create single-segment applications using CodeWarrior wizards, see [“Working With Palm OS Wizards” on page 38](#).

Refer to [“Working With Applications” on page 43](#) for information on working with Palm OS application projects using the CodeWarrior IDE.

Multi-Segment Applications

A *multi-segment application* is like a single-segment application except that the application contains multiple code segments. In a single-segment application, no code segment is larger than 64 kilobytes in size.

To learn how to use CodeWarrior wizards to create multi-segment applications, see [“Working With Palm OS Wizards” on page 38](#).

Refer to [“Working With Applications” on page 43](#) for information on working with Palm OS application projects using the CodeWarrior IDE.

Code Resources

A *code resource* is a Macintosh binary executable resource that adds functionality to a Palm OS application. Code resources cannot run by themselves. Instead, developers must include the binary in an application file and load it into memory before executing it from the application code.

Static Libraries

A *static library* contains functions that applications, code resources, and other libraries can use. Static libraries link together with the other components of the project and become part of the code within the application.

When you create a static library that uses other libraries, you do not need to add the other libraries to your project. If you instead use the `extern` keyword to declare the library symbols your static library uses as external, the CodeWarrior IDE links the static library without any errors.

TIP	If you add a static library that uses other libraries to an application project, you must also add the other libraries to the project.
------------	--

For more information, see [“Working With Libraries” on page 89](#).

NOTE	You must build static libraries with the same compiler settings as the applications that use them. For example, expanded mode settings and double / integer size settings should match. If you fail to match these settings you may encounter runtime problems such as mismatched program stacks and routines getting incorrect parameter values.
-------------	---

Shared Libraries

A *shared library* is a collection of routines that the Palm OS loads into memory separately from applications. Applications may call routines in a memory-resident shared library by using special system calls. Developers use shared libraries to implement common functions across a set of applications and to reduce memory usage.

You can create 68K shared libraries and ARM shared libraries with CodeWarrior Development Tools for Palm OS. For more information, see [“Working With Libraries” on page 89](#)

NOTE	Due to limited access to global variables, shared libraries have limited functionality.
-------------	---

For the Palm OS whitepaper on shared libraries, see: <http://oasis.palm.com/dev/kb/papers/1143.cfm>

Hacks

A *hack* is a file that the Palm OS loads separately using a shareware hack management program.

There are several hack management programs available for Palm OS:

- HackMaster <<http://www.daggerware.com/hackmstr.htm>>
- X-Master <<http://www.linkesoft.com/xmaster/>>
- TealMaster <<http://www.tealpoint.com/softmstr.htm>>

Hacks extend or modify aspects of the Palm OS by changing the way that applications work with the operating system, or changing aspects of the operating system itself.

For instance, a hack might change the way the operating system displays forms to show the time of day at the top of the screen. Another hack might perform some action whenever the user enters a special Graffiti character.

Writing hacks requires comprehensive knowledge of how the Palm OS works internally. Because hacks modify the operating system, you must be careful not to let them interfere with other software running in the operating system.

NOTE	Palm OS 5 does not support hacks due to changes in the operating system and the processors on Palm OS 5 devices.
-------------	--

For information about HackMaster, see this web page:

<http://www.daggerware.com/hackmstr.htm>

For information about creating hacks, see these web pages:

<http://www.daggerware.com/hackapi.htm>

<http://www.chronologic.se/Palm/sortinghack.html>

Examples of a CodeWarrior system hack projects are here (where *CWFolder* is the folder where you installed the CodeWarrior Development Tools for Palm OS package:

`CWFolder\ (CodeWarrior Examples)\Palm OS\Metrowerks
Examples\DemoHack`

Working With Palm OS Stationery

This section describes the Palm OS stationery included with the CodeWarrior Development Tools for Palm OS package.

CodeWarrior stationery projects allow you to create various types of CodeWarrior projects using the stationery as a foundation on which to build your project. There are several project stationery categories. Each is tailored towards a different type of Palm OS project, and contains appropriate settings, libraries, source code files, resource files, and segmentation information.

There are two parts to this section:

- [Stationery Reference](#)
- [Using Stationery](#)

Stationery Reference

These are the stationery projects you can use to create Palm OS projects:

- [Palm OS Application Stationery](#)
- [Palm OS ARMlet Stationery](#)
- [Palm OS Static Library Stationery](#)

Palm OS Application Stationery

Use the Palm OS application stationery to create Palm OS applications. [Table 2.1](#) describes the different types of application stationery available to you:

Table 2.1 Palm OS Application stationery descriptions

Stationery	Description
Palm OS C++ App	Use this stationery to create Palm OS application projects written in the C++ language.
Palm OS C App	Use this stationery to create Palm OS application projects written in the C language.
Palm OS Multi-Segment App	Use this stationery to create Palm OS multi-segment application projects written in the C language.

For detailed information about working with Palm OS application projects, see [“Working With Applications” on page 43](#).

Palm OS ARMLet Stationery

Use the Palm OS ARMLet stationery to create Palm OS application projects written in the C++ language that contain an ARMLet build target.

For detailed information about working with Palm OS application projects, see [“Working With Applications” on page 43](#).

For instructions about how to add ARMLets to existing application projects, read [“Adding ARMLets to Existing Projects” on page 58](#).

Palm OS Static Library Stationery

Use the Palm OS static library stationery to create Palm OS static libraries written in the C language.

For detailed information about working with Palm OS static library projects, see [“Working With Libraries” on page 89](#).

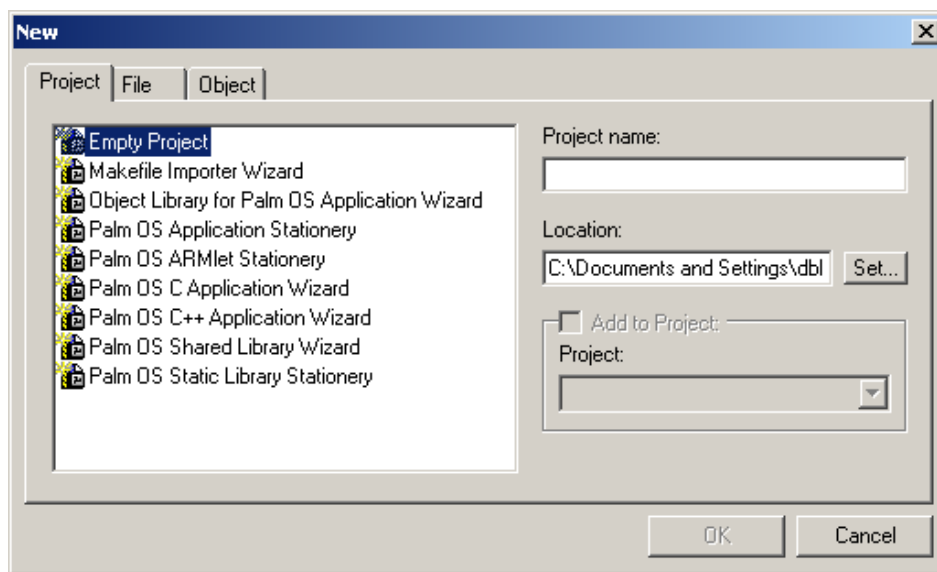
Using Stationery

Follow these steps to use stationery projects to create new CodeWarrior projects:

1. Run the CodeWarrior IDE
2. Choose **File > New** from the CodeWarrior menu bar.

The IDE displays the **New** window ([Figure 2.1](#)).

Figure 2.1 New window



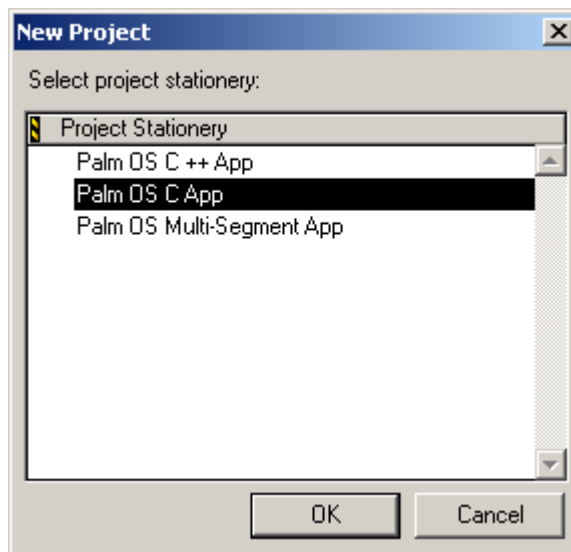
3. Select a stationery category from the list.
4. Enter a project name into the **Project Name** text field.

TIP The standard filename extension for CodeWarrior project files is .mcp.

5. Set the location for the project using the **Location** text field and the **Set** button.
6. Click the **OK** button.

If there are multiple stationery projects in this category, the IDE displays the **New Project** dialog box ([Figure 2.2](#)).

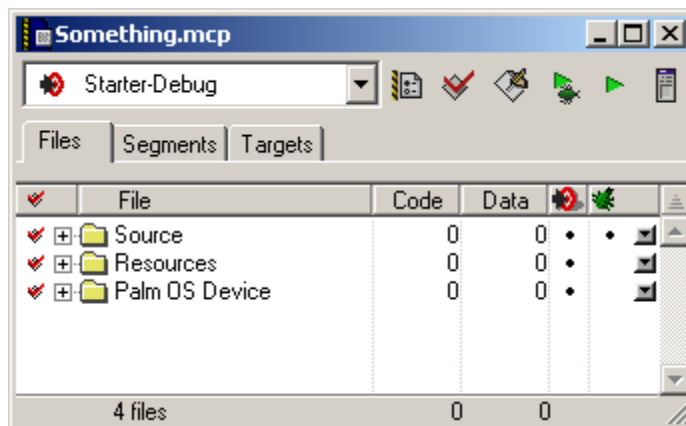
Figure 2.2 New Project dialog box



7. Select a stationery from the list.
8. Click **OK**.

The IDE creates the project and displays the Project window ([Figure 2.3](#)).

Figure 2.3 Project window



TIP You can create your own project stationery with your preferred settings. To create your own project stationery, modify a new project to suit your needs, then save it in the CodeWarrior (Project Stationery) folder. See the *IDE User's Guide* for more details.

Working With Palm OS Wizards

This section how to use the CodeWarrior wizards to create Palm OS projects. The CodeWarrior wizards allow you to easily create and configure CodeWarrior projects for Palm OS development. The wizards interactively create the project by using your input to generate projects with settings, libraries, source code files, resource files, and segmentation information tailored to your needs.

[Table 2.2](#) shows a description of each wizard in the CodeWarrior Development Tools for Palm OS package.

Table 2.2 Palm OS Wizards

Wizard	Description
Object Library for Palm OS Application Wizard	Use this wizard to create Palm OS applications written in the C++ language that use the Object Library for Palm OS C++ framework.
Palm OS C Application Wizard	Use this wizard to create Palm OS applications written in the C language.
Palm OS C++ Application Wizard	Use this wizard to create Palm OS applications written in the C++ language.
Palm OS Shared Library Wizard	Use this wizard to create Palm OS shared libraries written in the C language.

To learn more about specific wizards for creating Palm OS applications, see [“Creating Application Projects” on page 45](#).

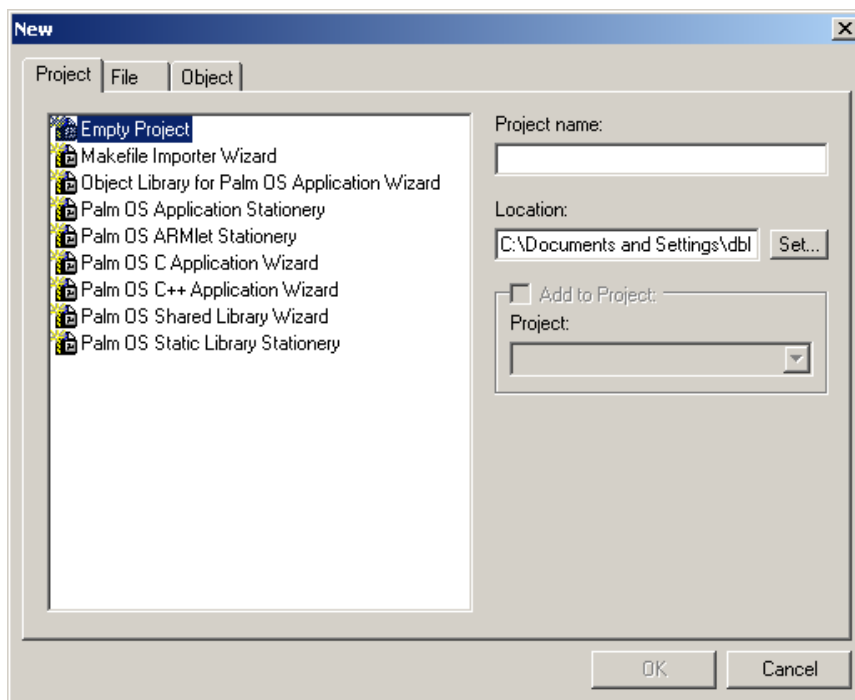
To learn more about the shared library wizard, refer to [“Working With Libraries” on page 89](#).

To use a wizard to create a CodeWarrior project for Palm OS:

1. Run the CodeWarrior IDE.
2. Select **File > New** from the CodeWarrior menu bar.

The IDE displays the **New** window ([Figure 2.4](#))

Figure 2.4 The New window



3. Select the wizard you want to use from the list.
4. Enter a name for the project into the **Project name** text field.

TIP The standard filename extension for CodeWarrior project files is .mcp.

5. Click the **Set** button to set the location for the project folder.
6. Click **OK** to start the wizard.
7. Follow the instructions the wizard presents.

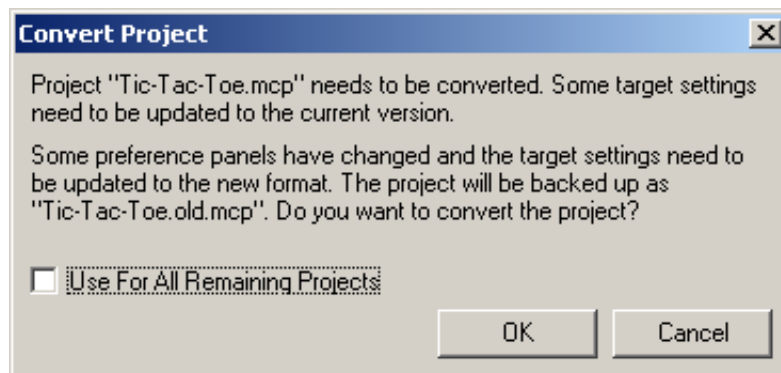
Converting Older Projects to Version 9

This section describes how to convert CodeWarrior Development Tools for Palm OS version 8 projects to CodeWarrior Development Tools for Palm OS version 9 projects.

To upgrade your projects to the new version of the CodeWarrior tools:

1. Make a backup copy of the project and associated source files.
2. Run the CodeWarrior IDE.
3. Select **File > Open** from the CodeWarrior menu bar and open the project file you want to convert.
4. The IDE displays the **Convert Project** dialog box ([Figure 2.5](#)).

Figure 2.5 The Convert Project dialog box



5. Click **OK**,
The IDE converts the target settings in the project to the latest version and displays the Project window. The IDE also displays a **Project Messages** window listing the target settings panel data that it updates to the latest version.
6. Select **Edit > TargetName Settings** (where *TargetName* is the name of the build target you want to change).
The IDE displays the **Target Settings** window.
7. Select **Access Paths** from the list on the left side of the **Target Settings** window.
The IDE displays the **Access Paths** settings panel.

8. Delete these paths from the system paths list:

```
{Compiler}Palm OS 3.1 Support  
{Compiler}Palm VII Support  
{Compiler}Palm OS 3.5 Support  
{Compiler}(MSL for Palm OS)
```

9. Add these paths to the system path list, in this order:

```
{Compiler}Palm OS Support  
{Compiler}CW for Palm OS Support
```

10. Delete any SDK paths (where *SDK-Name* is the name of an SDK):

```
{Compiler}Other SDKs\SDK-Name
```

Replace them with:

```
{Compiler}CW for Palm OS Support\ (Other SDKs)\SDK-Name
```

11. If the project used the Palm OS 3.1 or VII SDK, search for this path in all project source files:

```
#include <Pilot.h>
```

Replace it with these paths:

```
#include <PalmOS.h>
```

```
#include <PalmCompatibility.h>
```

12. If the project used MSL libraries, switch to the new MSL for Palm OS.

See [“MSL Libraries” on page 101](#) for details.

Working With Projects

Converting Older Projects to Version 9

Working With Applications

This chapter provides helpful information about working with application projects using the CodeWarrior IDE.

The sections in this chapter are:

- [Types of Application Projects](#) — describes the different types of applications projects you can create with the CodeWarrior IDE
- [Creating Application Projects](#) — describes the various wizards you can use to create different types of Palm OS application projects
- [Single-Segment Application Limitations](#) — describes the Palm OS® 16-bit reference limitation affecting single-segment applications and provides some methods to work around the limitation
- [Converting Single-Segment Applications to Multi-Segment Applications](#) — shows how to use the CodeWarrior IDE to convert a single-segment application project into a multi-segment application project
- [Adding ARMlets to Existing Projects](#) — describes how to add ARMlets to existing CodeWarrior application projects

NOTE

The *CodeWarrior Tutorial for Palm OS* book provides a detailed tutorial that shows you how to create a Palm OS application using the latest and greatest features of the CodeWarrior Development Tools for Palm OS package. This book is located in the following folder (where *CWFolder* is the CodeWarrior installation folder):

```
CWFolder\CodeWarrior Manuals\MW Tutorial for Palm OS\
```

Types of Application Projects

This section describes the different types of applications you can create with the CodeWarrior IDE. There are two basic application types for Palm OS:

- [Single-Segment Applications](#)
- [Multi-Segment Applications](#)

Single-Segment Applications

A *single-segment application* is a Palm OS application that contains a single code segment that is smaller than 64 kilobytes in size. This is the most common type of application on the Palm OS.

For a list of the CodeWarrior runtime libraries you need to create applications, see [“Runtime Libraries” on page 98](#).

[“Single-Segment Application Limitations” on page 55](#) describes the Palm OS® 16-bit reference limitation affecting single-segment applications and provides some methods to work around the limitation.

To learn how to create a single-segment application, see [“Creating Application Projects” on page 45](#).

Multi-Segment Applications

A *multi-segment application* is like a single-segment application except that the application contains multiple code segments. In a single-segment application, no code segment is larger than 64 kilobytes in size.

For a list of the CodeWarrior runtime libraries you need to create applications, see [“Runtime Libraries” on page 98](#).

To learn how to convert single-segment applications to multi-segment applications, see [“Converting Single-Segment Applications to Multi-Segment Applications” on page 57](#).

Creating Application Projects

This section shows you how to create application projects using CodeWarrior application wizards. The CodeWarrior application wizards interactively create the project by using your input to generate projects with settings, libraries, source code files, resource files, and segmentation information tailored to your needs

To use one of the application wizards to create a Palm OS project:

1. Run the CodeWarrior IDE.
2. Select **File > New** from the CodeWarrior menu bar.
The IDE displays the **New** window.
3. Select the wizard you want to use.
4. Enter a name for the project into the **Project name** text field.

TIP	The standard filename extension for CodeWarrior project files is .mcp.
------------	---

5. Click the **Set** button to set the location for the project folder.
6. Click **OK** to start the wizard.
7. Follow the instructions the wizard presents.

The rest of this section describes the individual application wizards included with the CodeWarrior Development Tools for Palm OS package.

These are the included wizards:

- [Object Library for Palm OS Application Wizard](#)
- [Palm OS C and C++ Application Wizards](#)

Object Library for Palm OS Application Wizard

Use the Object Library for Palm OS Application wizard to create Palm OS applications written in the C++ language that use the Object Library for Palm OS (POL) C++ framework.

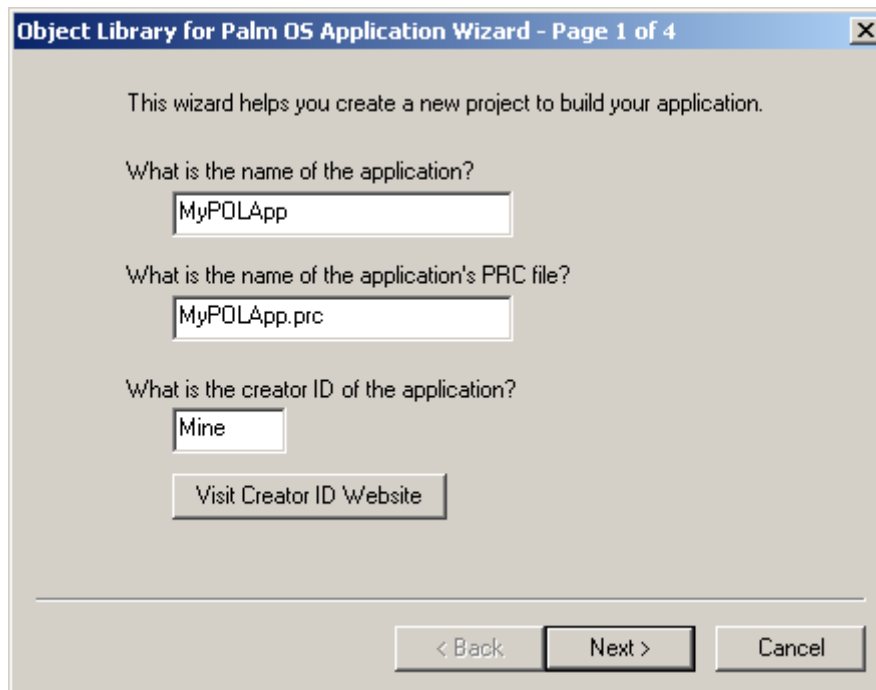
The Object Library for Palm OS C++ framework is a library of C++ classes designed for the Palm OS platform by AQPoint Consulting. For detailed information about POL, point your web browser to:

<http://www.aqpoint.com/pol/>

Page 1 of 4

The first page of the wizard ([Figure 3.1](#)) asks you for information about the application you want to create.

Figure 3.1 Object Library for Palm OS Wizard - Page 1 of 4



Object Library for Palm OS Application Wizard - Page 1 of 4

This wizard helps you create a new project to build your application.

What is the name of the application?

What is the name of the application's PRC file?

What is the creator ID of the application?

Enter the name of the application. This is the name by which users identify the application on the Palm OS device.

Enter the name of the application PRC file. This is the name of the PRC file the CodeWarrior IDE generates when you build the project.

NOTE Developers sometimes use a shortened or abbreviated version of the application name for the PRC file name.

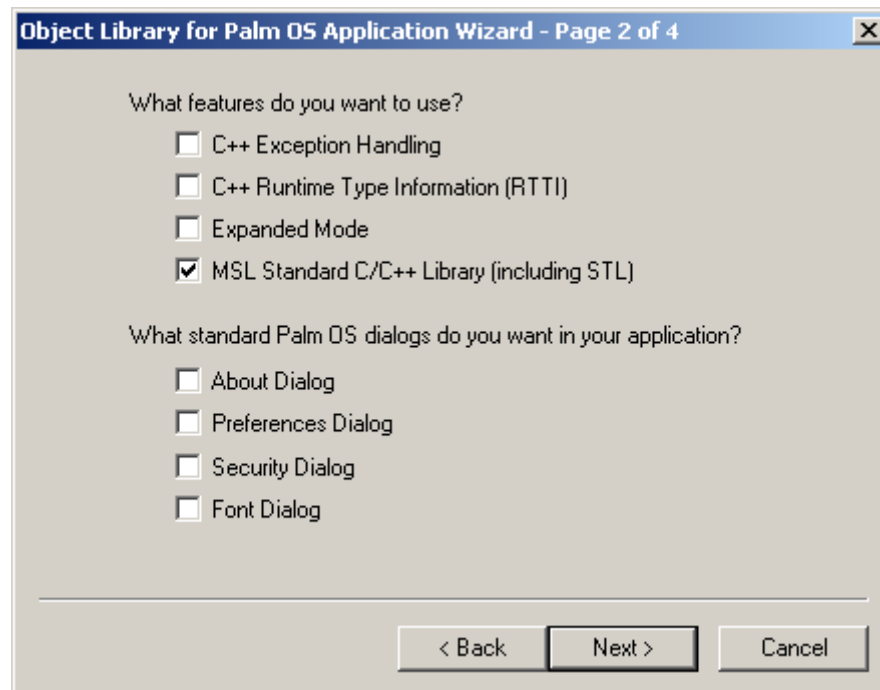
Enter the creator code for the application. The creator code should be a four-character code that is unique to your application. Click the **Visit Creator ID Website** button to view the Palm OS developer creator ID web site in your web browser.

NOTE You should ensure that the creator ID you choose is unique and is registered with the Palm OS creator ID website before you make the application available to the public.

Page 2 of 4

On this page ([Figure 3.2](#)), the wizard asks you to choose the C++ language features and standard dialogs you want to use in the application.

Figure 3.2 Object Library for Palm OS Wizard - Page 2 of 4



Check the **C++ Exception Handling** box to enable C++ exception handling in the application. When you check this box, the IDE enables the **Enable C++ Exceptions** setting in the **C/C++ Language** target settings panel.

Check the **C++ Runtime Type Information (RTTI)** box to enable support for runtime type information (RTTI), including the `dynamic_cast` and `typeid` operators. When you check this box, the IDE enables the **Enable RTTI** setting in the **C/C++ Language** target settings panel.

Check the **Expanded Mode** box to have the application use expanded mode. Expanded mode reduces the amount of data stored in global space for the application by storing compiler-generated data such as C++ virtual tables, multi-segment jump tables, exception handling information, and RTTI info in database RAM where programs and data files are stored. This leaves you more global space for the application. See [“Expanded Global Space Mode” on page 198](#) for more details.

Check the **MSL Standard C/C++ Library (including STL)** box to have the wizard add the appropriate access paths and libraries to the project to support standard MSL C and C++ functionality.

Check the **About Dialog** box to have the wizard generate an **About Application** menu item and corresponding dialog box.

Check the **Preferences Dialog** box to have the wizard generate a **Preferences** menu item and corresponding Preferences dialog box that users can use to set application preferences.

Check the **Security Dialog** box to have the wizard generate a **Security** menu item that generates a call to Palm OS to let users set record privacy settings.

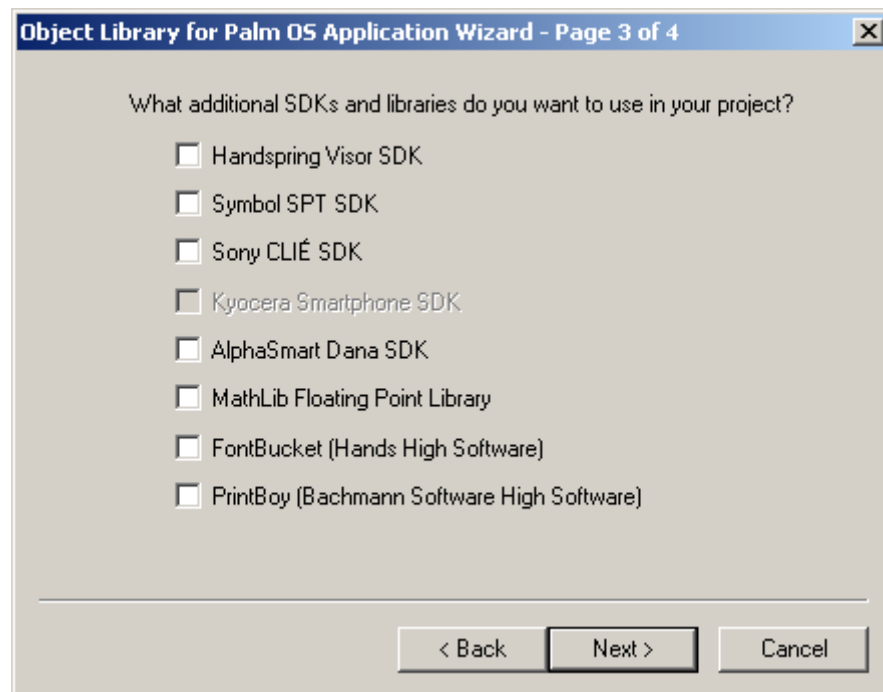
Check the **Font Dialog** box to have the wizard generate a **Font** menu item that generates calls to Palm OS to show the font selection dialog.

TIP	If you check this box and also check the FontBucket box on Page 3 of 4 of the wizard, the wizard generates code that calls into the FontBucket library if it is available on the host device.
------------	--

Page 3 of 4

On this page ([Figure 3.3](#)), the wizard asks you to choose the software development kits (SDKs) and libraries you want to use in the project.

Figure 3.3 Object Library for Palm OS Wizard - Page 3 of 4



Check the boxes next to the SDKs and libraries you want to use in your application.

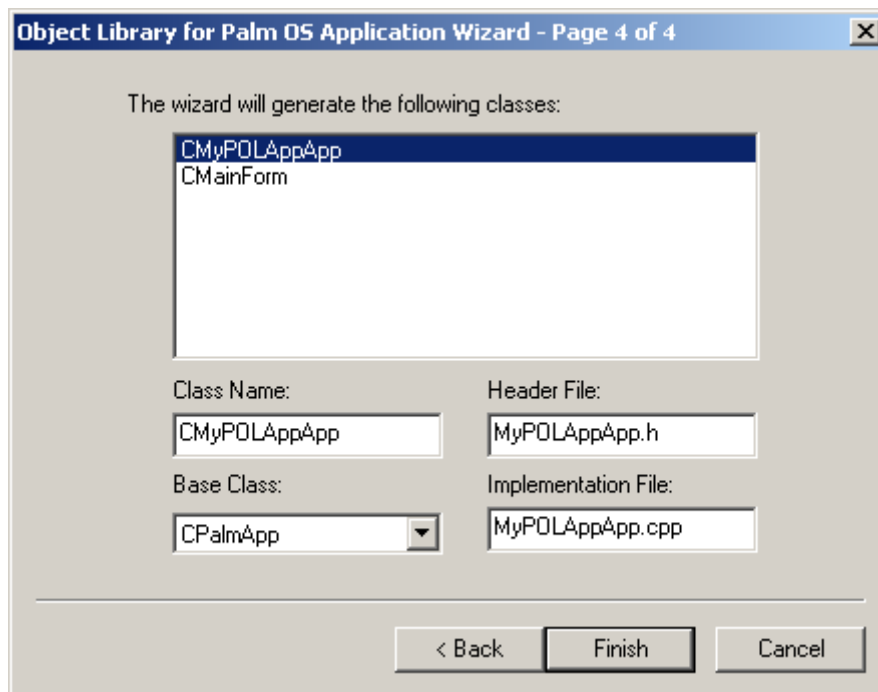
NOTE	The list of SDKs you see in this page of the wizard may be different in your particular installation of CodeWarrior Development Tools for Palm OS.
-------------	--

TIP	If you check the FontBucket box and also checked the Font Dialog box on Page 2 of 4 of the wizard, the wizard generates code that calls into the FontBucket library if it is available on the host device.
------------	--

Page 4 of 4

On this page ([Figure 3.4](#)), the wizard allows you to change properties of the POL classes the wizard generates when it creates your project.

Figure 3.4 Object Library for Palm OS Wizard - Page 4 of 4



Select a class from the class list to view or change its properties. The wizard updates the values of the rest of the items on the page to reflect your selection.

Enter in the **Class Name** text field the name you want the wizard to give the class. The wizard uses this name to identify the class in the source code it generates for the class.

Select an item from the Base Class menu to specify the base class for the class.

Enter in the **Header File** text field the name for the class header file. Header file names typically have the .h extension.

Enter in the **Implementation File** text field the name of the class implementation file. Implementation file names typically have the .cpp extension.

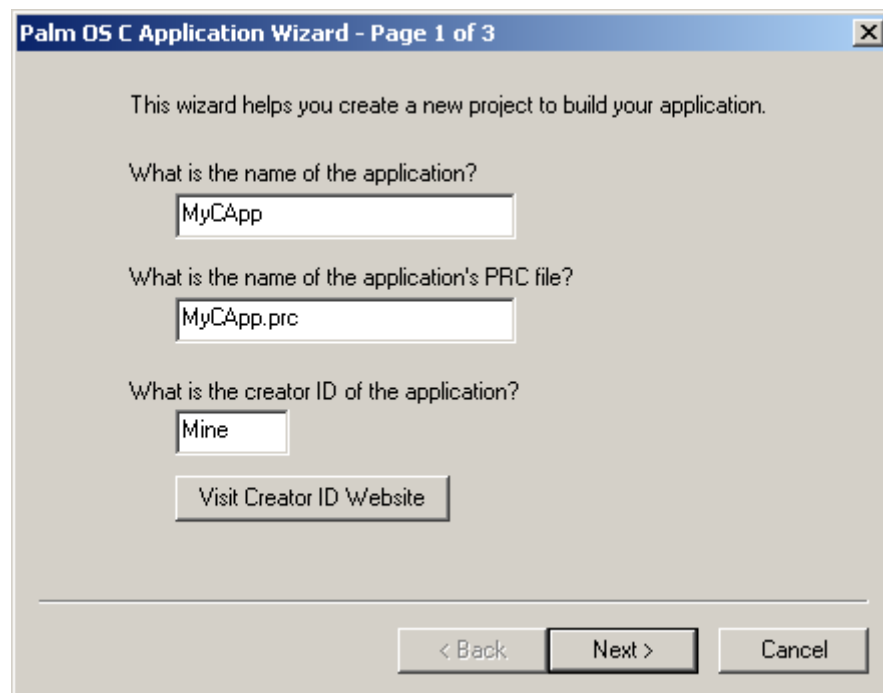
Palm OS C and C++ Application Wizards

Use the Palm OS C and C++ Application wizards to create Palm OS applications written in the C or C++ languages.

Page 1 of 3

The first page of the wizard ([Figure 3.5](#)) asks you for information about the application you want to create.

Figure 3.5 Palm OS C and C++ Application wizards - Page 1 of 3



Enter the name of the application. This is the name by which users identify the application on the Palm OS device.

Enter the name of the application PRC file. This is the name of the PRC file the CodeWarrior IDE generates when you build the project.

NOTE Typically, developers use a shortened or abbreviated version of the application name for the PRC file name.

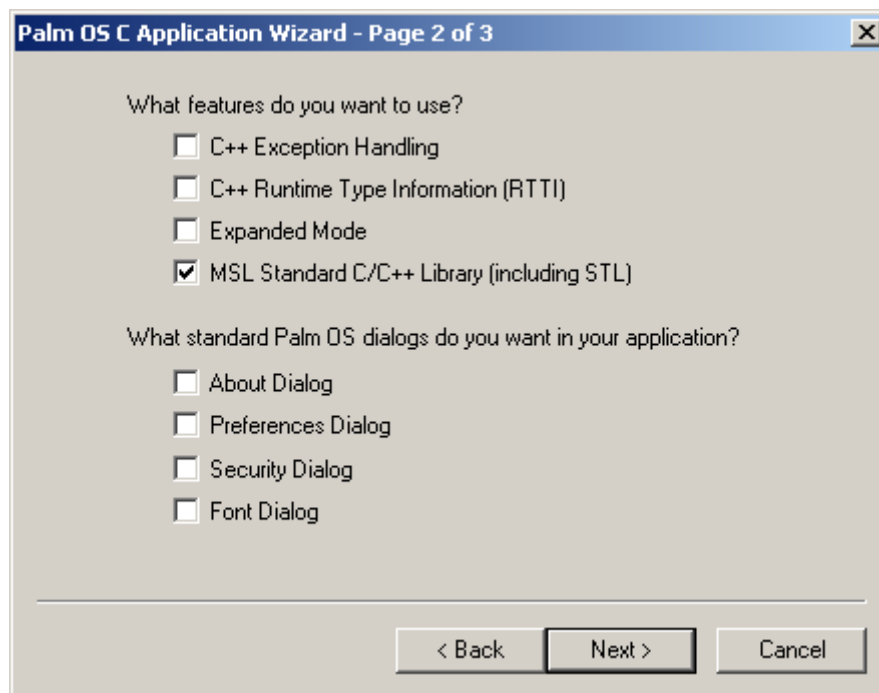
Enter the creator code for the application. The creator code should be a four-character code that is unique to your application. Click the **Visit Creator ID Website** button to view the Palm OS developer creator ID web site in your web browser.

NOTE You should ensure that the creator ID you choose for your application is unique and is registered with the Palm OS creator ID website before you make the application available to the public.

Page 2 of 3

On this page ([Figure 3.6](#)), the wizard asks you to choose the C++ language features and standard dialogs you want to use in the application.

Figure 3.6 Palm OS C and C++ Application wizards - Page 2 of 3



Check the **C++ Exception Handling** box to enable C++ exception handling in the application. When you check this box, the IDE enables the **Enable C++ Exceptions** setting in the **C/C++ Language** target settings panel.

Check the **C++ Runtime Type Information (RTTI)** box to enable support for runtime type information (RTTI), including the `dynamic_cast` and `typeid`

operators. When you check this box, the IDE enables the **Enable RTTI** setting in the **C/C++ Language** target settings panel.

Check the **Expanded Mode** box to have the application use expanded mode. Expanded mode reduces the amount of data stored in global space for the application by storing compiler-generated data such as C++ virtual tables, multi-segment jump tables, exception handling information, and RTTI info in database RAM where programs and data files are stored. This leaves you more global space for the application. See [“Expanded Global Space Mode” on page 198](#) for more details.

Check the **MSL Standard C/C++ Library (including STL)** box to have the wizard add the appropriate access paths and libraries to the project to support standard MSL C and C++ functionality.

Check the **About Dialog** box to have the wizard generate an **About Application** menu item and corresponding dialog box.

Check the **Preferences Dialog** box to have the wizard generate a **Preferences** menu item and corresponding Preferences dialog box that users can use to set application preferences.

Check the **Security Dialog** box to have the wizard generate a **Security** menu item that generates a call to Palm OS to let users set record privacy settings.

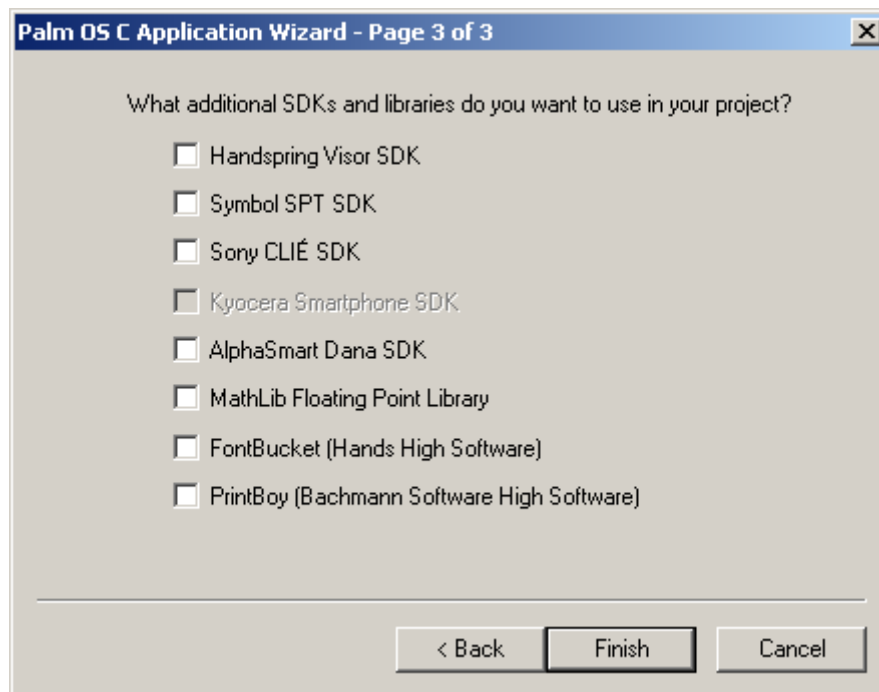
Check the **Font Dialog** box to have the wizard generate a **Font** menu item that generates calls to Palm OS to show the font selection dialog.

TIP	If you check this box and also check the FontBucket box on Page 3 of 4 of the wizard, the wizard generates code that calls into the FontBucket library if it is available on the host device.
------------	--

Page 3 of 3

On this page ([Figure 3.7](#)), the wizard asks you to choose the software development kits (SDKs) and libraries you want to use in the project.

Figure 3.7 Palm OS C and C++ Application wizards - Page 3 of 3



Check the boxes next to the SDKs and libraries you want to use in your application.

NOTE	The list of SDKs you see in this page of the wizard may be different in your particular installation of CodeWarrior Development Tools for Palm OS.
-------------	--

TIP	If you check the FontBucket box and also checked the Font Dialog box on Page 2 of 3 of the wizard, the wizard generates code that calls into the FontBucket library if it is available on the host device.
------------	--

Single-Segment Application Limitations

The design of the Palm OS limits jumps to a maximum distance of 32KB. This section provides some ways to work around this limitation.

If the application size is greater than 32K bytes, the linker may report the error message “16-bit reference out of range” when attempting to build the project. This error means that a routine is referring to another routine more than 32K bytes away.

Here are some methods for working around the problem:

- [Changing the link order of the application](#)
- [Using the Smart code model](#)
- [Reorganizing or rewriting the application](#)
- [Using Jump Islands](#)

Changing the link order of the application

In many cases, you can reorganize the location of routines within the final application to solve the 16-bit jump limitation problem. If the routines are in separate project files, you can change the location of the object code by changing the order of the project files in the **Segments** pane of the Project window.

Note, however, that as you continue to add code to the application making the application larger, it may not be possible to reorganize the source code to solve the size limitation problem.

To change the link order:

1. Refer to each “16-bit reference out of range” linker error message to determine which routine in your code is attempting to call another routine more than 32K bytes away.
2. Determine the source files that contain the calling function and target function.
3. Move those source files closer together in the **Segments** pane of the Project window.

Using the Smart code model

You can attempt to resolve out-of-range function references by instructing the compiler and linker to use 32-bit jump instructions rather than standard 16-bit jump instructions. Note, however, that doing so causes the compiler and linker to simulate each 32-bit jump by adding many extra instructions to the code. To use the smart code model:

1. Select **Edit > TargetName Settings** from the CodeWarrior menu bar (where *TargetName* is the name of the build target you want to change).

The IDE displays the Target Settings window.

2. Click **68K Processor** in the **Target Settings Panels** list on the left side of the Target Settings window.

The IDE displays the **68K Processor** settings panel.

3. Select **Smart** from the **Code Model** menu.

The IDE changes the code model to smart.

Reorganizing or rewriting the application

You can rewrite your code to resolve out-of-range function references by grouping functions that call each other so that they are closer together in their own source code files.

You can also convert your application to a multi-segment application. See [“Converting Single-Segment Applications to Multi-Segment Applications” on page 57](#) for instructions on how to do this.

Using Jump Islands

You may use jump islands and jump tables to avoid the 32 KB jump limitation. For more information, refer to the following Palm, Inc. document:

<http://oasis.palm.com/dev/kb/faq/1418.cfm>

To learn about converting a single-segment application into a multi-segment application, see [“Converting Single-Segment Applications to Multi-Segment Applications” on page 57](#).

For information on how the C and C++ compilers generate object code, see [“C and C++ Compilers” on page 177](#).

Converting Single-Segment Applications to Multi-Segment Applications

To convert a single-segment application project into a multi-segment application project:

1. Create a new build target for the multi-segment application.

In your Palm OS application project, create a new build target for the multi-segment application, copied from the single-segment application target. Creating the multi-segment application target this way ensures that you use the same files and settings used by the single-segment application.

2. Change the multi-segment application build target settings.

Uncheck the **Link Single Segment** checkbox in the **68K Linker** settings panel. For more information on this settings panel, see [“68K Linker” on page 166](#).

3. Replace the runtime libraries.

Multi-segment applications use different runtime libraries and files than single-segment applications. See [“Runtime Libraries” on page 98](#) for more information.

4. Rearrange the link order in the **Segments** view of the Project window.

Multi-segment applications require that runtime libraries and other startup code be in the first group of the **Segment** view of the Project window. See [“Multi-Segment Project Libraries” on page 99](#) for more information.

Adding ARMlets to Existing Projects

This section describes how to add an ARMlet to an existing CodeWarrior application project.

An ARMlet is a self-contained ARM-native program that is called from within a 68K application running on Palm OS 5. ARMlets run natively on ARM processors rather than running through an emulator like 68K code. Developers typically store ARMlet executable code in application resources with a resource type code of 'ARMC', which stands for "ARM Code". The Palm OS does not enforce the resource type code or location, however; so ARMlets may be stored elsewhere and have a different creator type codes.

TIP	To learn more about ARMlet programming, see the book <i>Palm OS Companion</i> included in the Palm OS SDK 5.0.
------------	--

You must complete these tasks sequentially:

- [Open the Project](#)
- [Create a New Build Target](#)
- [Configure the Build Target Linker](#)
- [Configure the Build Target Access Paths](#)
- [Add the ARMlet Runtime Library](#)
- [Write ARMlet Source Code](#)
- [Add ARMlet Source Code to the Project](#)
- [Create Build Target Dependencies](#)
- [Merge the ARMlet Executable Code Into the Application](#)
- [Build the Application Target](#)

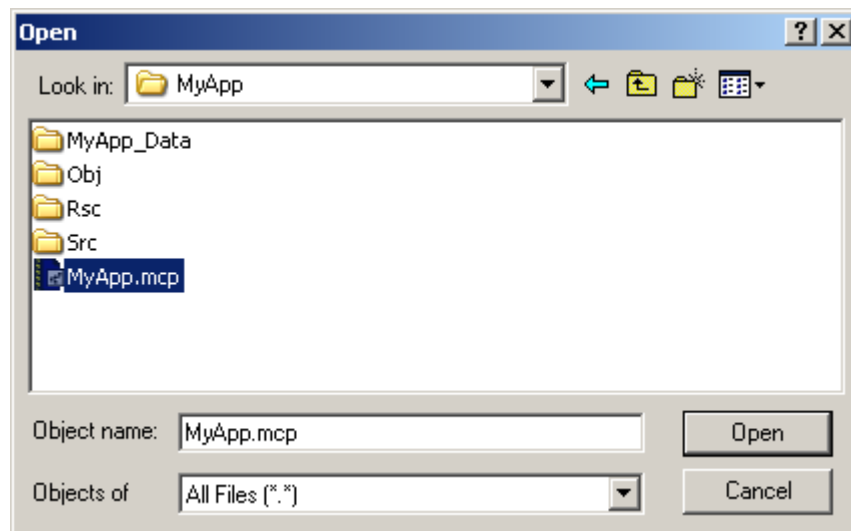
Open the Project

First, open the project to which you want to add the ARMLet. The project should contain at least one build target that generates a Palm OS 68K application.

1. Run the CodeWarrior IDE.
2. Select **File > Open** from the CodeWarrior menu bar.

The CodeWarrior IDE displays the **Open** dialog box ([Figure 3.8](#)).

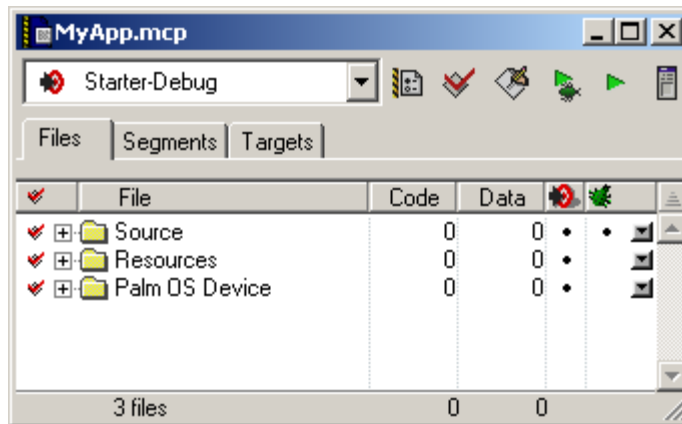
Figure 3.8 Open dialog box



3. Open the project to which you want to add the ARMlet.

The IDE displays the Project window ([Figure 3.9](#)).

Figure 3.9 Project window



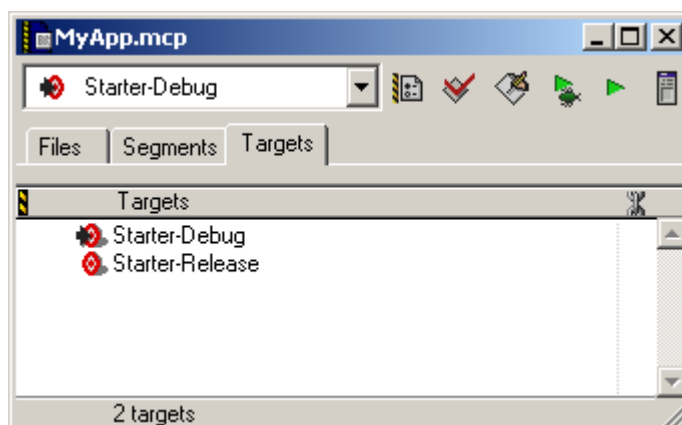
Create a New Build Target

Next, add a new build target to the project. This build target will contain the settings and source files that the IDE will use to generate the ARMlet.

1. Click the **Targets** tab in the Project window.

The IDE displays the **Targets** page of the Project window ([Figure 3.10](#)).

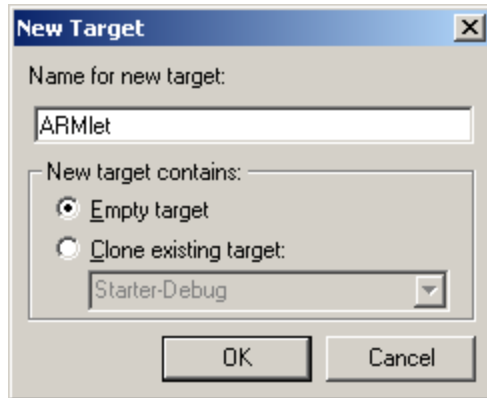
Figure 3.10 Targets page of the Project window



2. Select **Project > Create Target** from the CodeWarrior menu bar.

The IDE displays the **New Target** window ([Figure 3.11](#)).

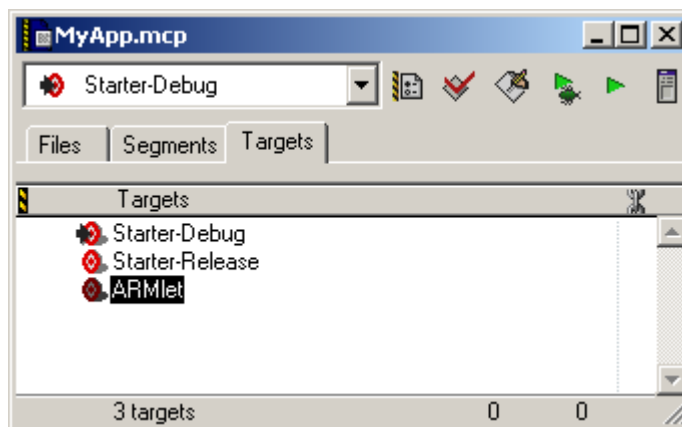
Figure 3.11 New Target dialog box



3. Enter a name for the ARMLet build target in the **Name for new target** text field (for example ARMLet).
4. Check the **Empty target** box to signify that you want the IDE to create an empty build target.
5. Click **OK**.

The IDE creates an empty build target with the name you specified and updates the **Targets** page of the Project window ([Figure 3.12](#)).

Figure 3.12 Project window showing the new build target



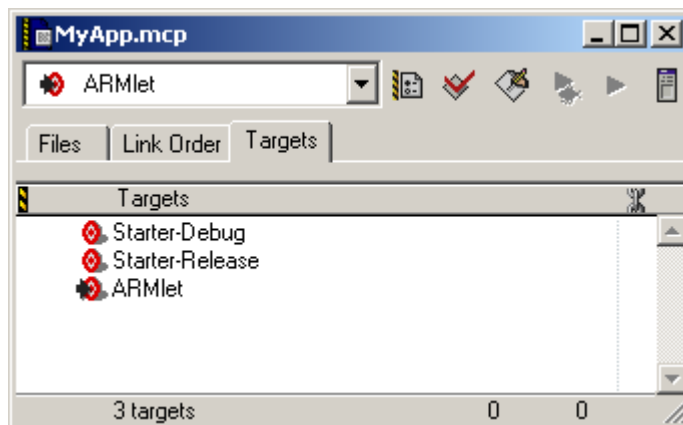
Configure the Build Target Linker

Next, configure the new build target to use the Palm OS ARMlet compiler and linker to generate the ARMlet. You will also set various attributes of the final ARMlet binary executable in this section.

1. Select **Project > Set Default Target > *TargetName*** from the CodeWarrior menu bar (where *TargetName* is the name of the new build target).

The IDE switches to the new build target (as shown in [Figure 3.13](#)).

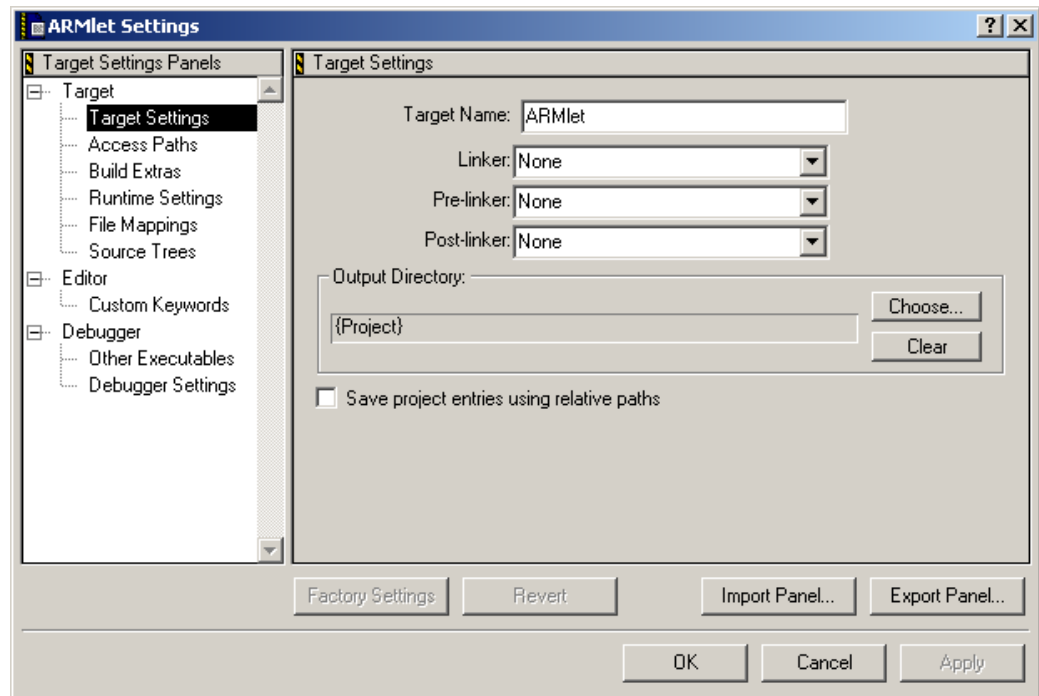
Figure 3.13 Project window - new build target selected



2. Select **Edit > TargetName Settings** from the CodeWarrior menu bar (where *TargetName* is the name of the new build target).

The IDE displays the **Target Settings** window ([Figure 3.14](#)).

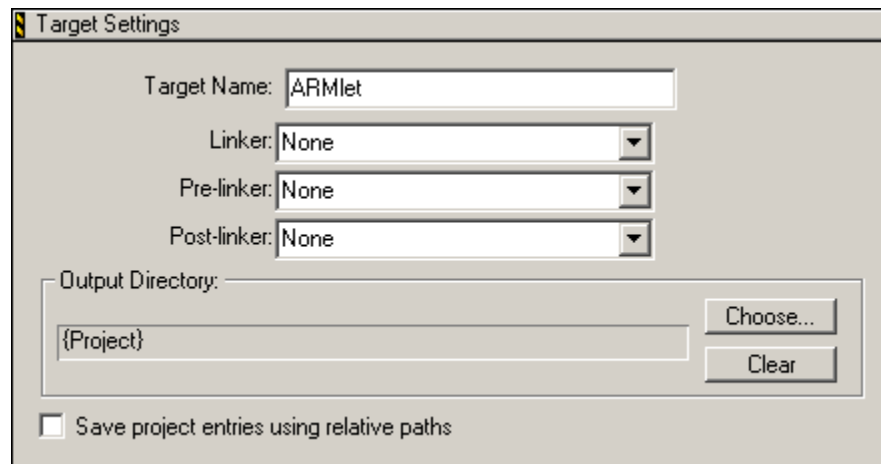
Figure 3.14 Target Settings window



3. Select **Target Settings** from the **Target Settings Panels** list on the left side of the Target Settings window.

The IDE displays the **Target Settings** panel ([Figure 3.15](#)).

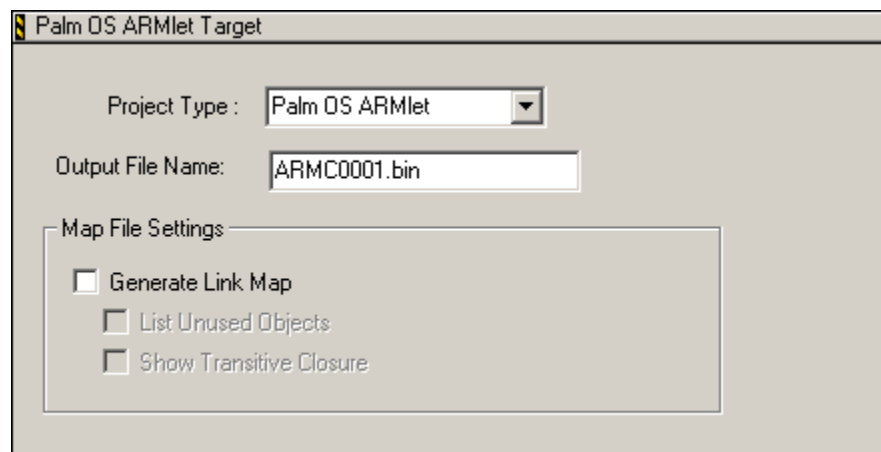
Figure 3.15 Target Settings panel



4. Select **Palm OS ARMlet** from the **Linker** menu in the Target Settings panel.
The IDE switches to the **Palm OS ARMlet** linker.
5. Select **Palm OS ARMlet Target** from the **Target Settings Panels** list on the left side of the Target Settings window.

The IDE displays the **Palm OS ARMlet Target** settings panel ([Figure 3.16](#)).

Figure 3.16 Palm OS ARMlet settings panel



6. Enter in the **Output File Name** text field the name of the ARMlet binary executable file.

The name you enter here should end with the filename extension `.bin` and should be based on the resource type and ID you want the ARMlet resource to have. The first four characters should correspond to the resource type code. The next four characters should correspond to the hexadecimal representation of the resource ID.

For example, if you want the ARMlet resource to have the resource type code 'ARMC', and to have the resource ID 1000, set the name to `ARMC03E8.bin`.

TIP	You may add additional text to the filename after the first eight characters if you wish - the linker ignores everything after the first eight characters.
------------	--

7. Click **Apply** in the **Target Settings** window.

The IDE saves your changes.

The new build target is now configured to create an ARMlet binary code resource the next time you build it.

Configure the Build Target Access Paths

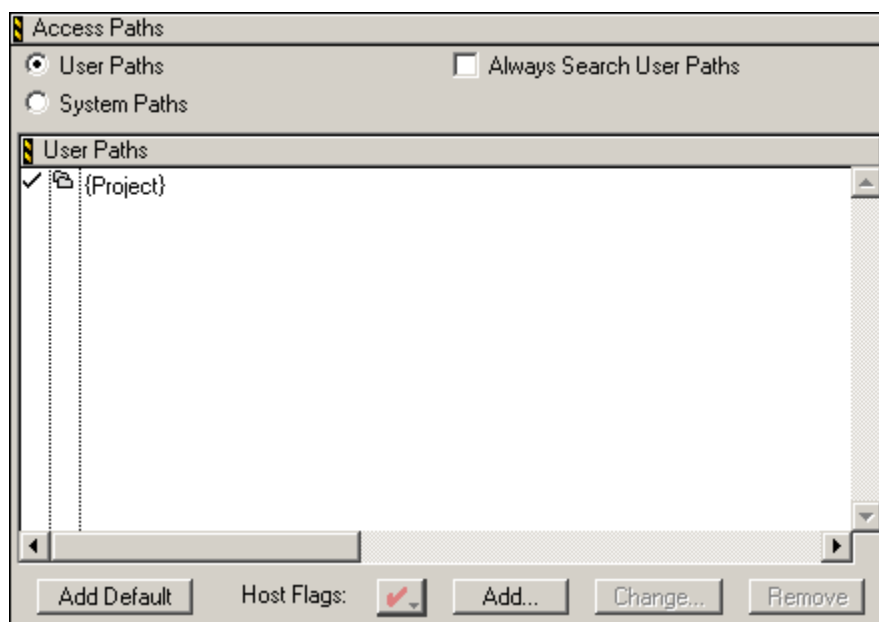
Next, configure the access paths the ARMlet compiler and linker use to find project header files.

You will change one access path and add two more access paths so that the Palm OS ARMlet compiler can find the appropriate ARMlet support files.

1. Select **Access Paths** in the Target Settings Panels list on the left side of the Target Settings window.

The IDE displays the **Access Paths** settings panel ([Figure 3.17](#)).

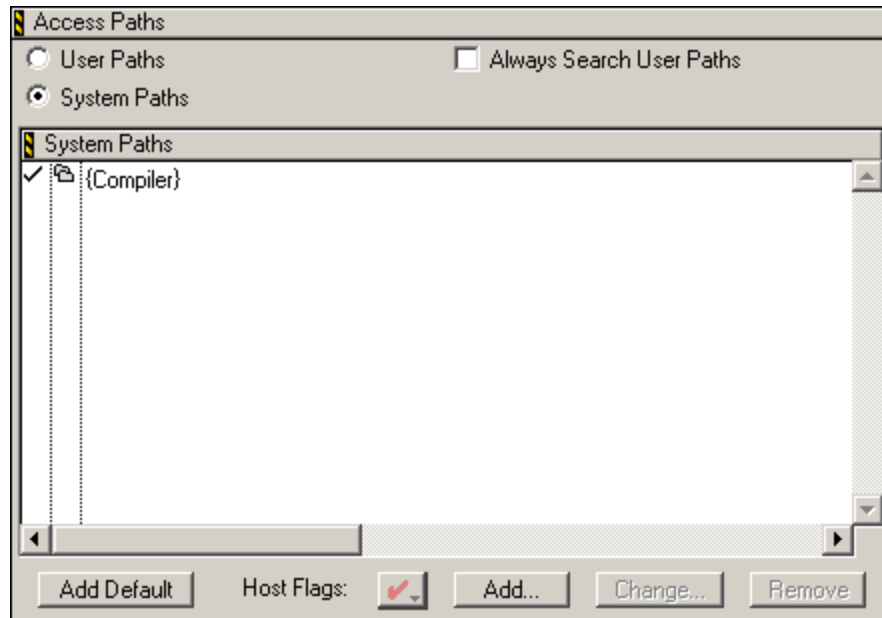
Figure 3.17 Access Paths settings panel: User Paths list



2. Click the **System Paths** option.

The IDE displays the system paths list ([Figure 3.18](#)).

Figure 3.18 Access Paths settings panel: System Paths list



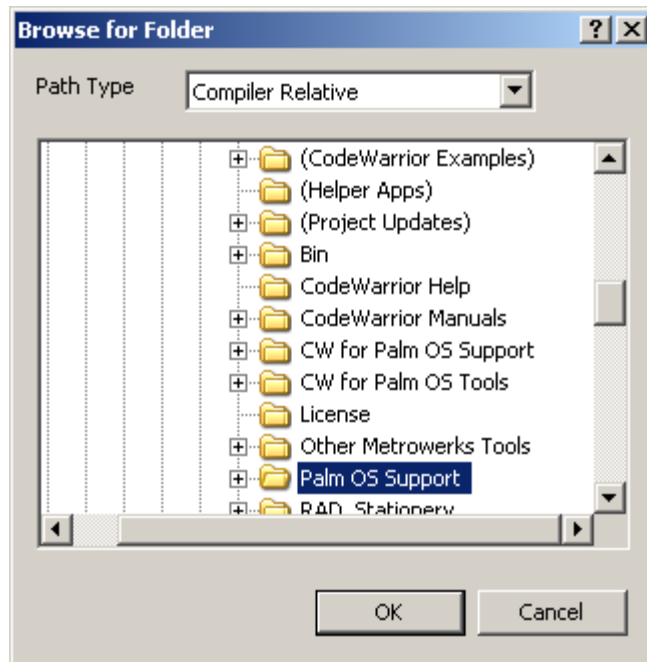
3. Select the **{Compiler}** access path.

The IDE highlights your selection.

4. Click **Change**.

The IDE displays the **Browse for Folder** dialog box ([Figure 3.19](#)).

Figure 3.19 Browse for Folder dialog box



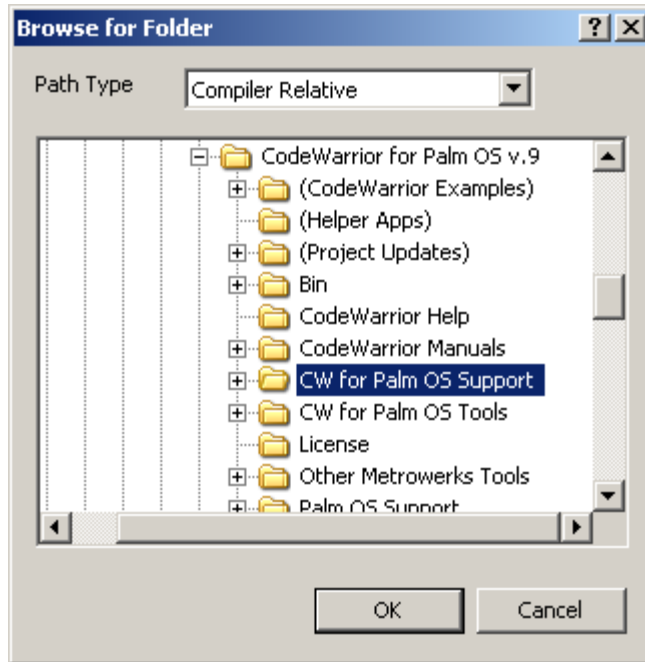
5. Select **Compiler Relative** from the **Path Type** menu.
6. Locate the `Palm OS Support` folder within the CodeWarrior Development Tools for Palm OS installation folder and select it.
7. Click **OK**.

The IDE closes the **Browse for Folder** dialog box and updates the path in the system path list.

8. Click **Add**.

The IDE displays the **Browse for Folder** dialog box ([Figure 3.20](#)).

Figure 3.20 Browse for Folder dialog box

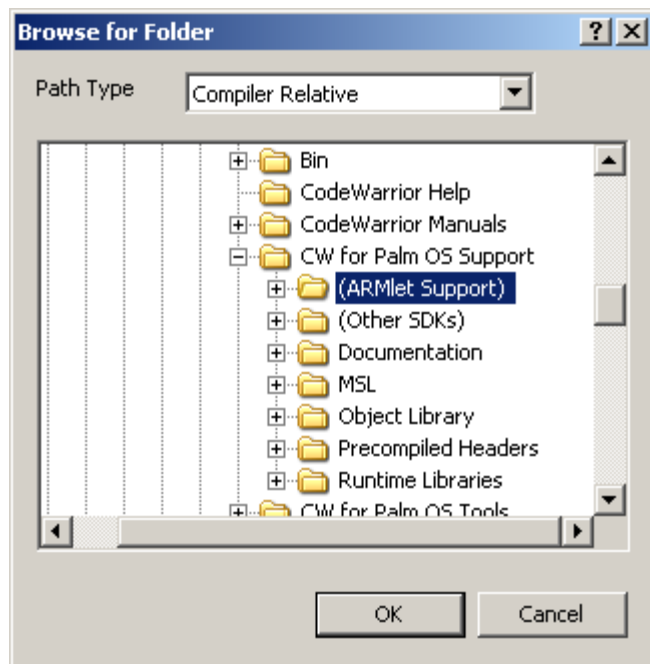


9. Select **Compiler Relative** from the **Path Type** menu.
10. Locate the CW for Palm OS Support folder within the CodeWarrior Development Tools for Palm OS installation folder and select it.

11. Click **Add**.

The IDE displays the **Browse for Folder** dialog box ([Figure 3.21](#)).

Figure 3.21 Browse for Folder dialog box

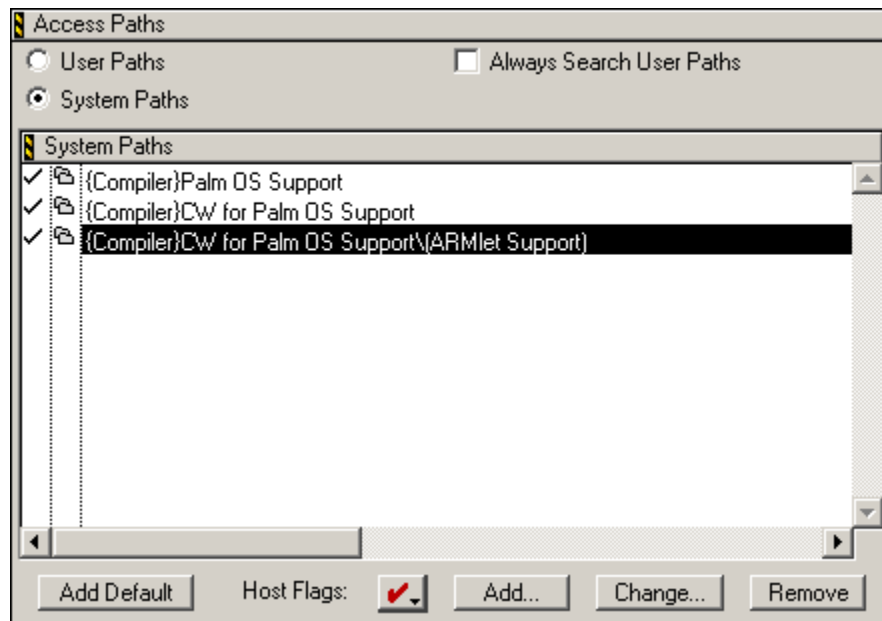


12. Select **Compiler Relative** from the **Path Type** menu.
13. Locate the CW for Palm OS Support\ (ARMlet Support) folder within the CodeWarrior Development Tools for Palm OS installation folder and select it.

14. Click **OK**.

The IDE closes the **Browse for Folder** dialog box and updates the path in the system path list ([Figure 3.22](#)).

Figure 3.22 Access Paths settings panel



15. Click **OK** in the **Target Settings** window.

The IDE saves your changes and closes the **Target Settings** window.

The ARMlet build target access paths are now configured so that the next time you build this target the compiler will be able to locate the ARMlet support files.

Add the ARMlet Runtime Library

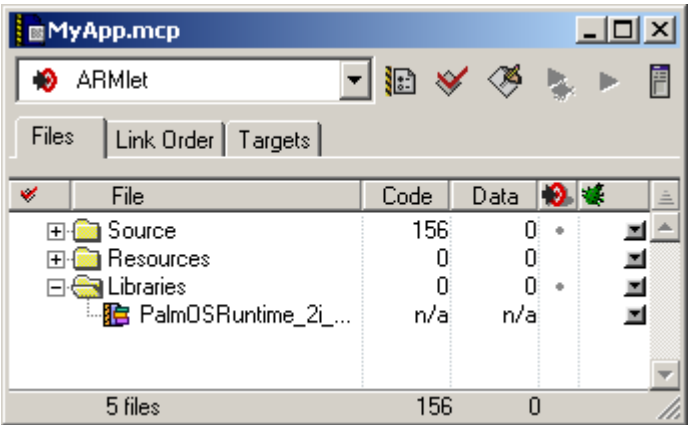
Next, add the ARMlet runtime library file to the build target.

The ARMlet runtime library contains the runtime startup code that allows the ARMlet to run on Palm OS. The Palm OS ARMlet compiler and linker add this runtime library code to the ARMlet when you build the ARMlet target.

1. In the Project window, click the **Files** tab.

The IDE displays the **Files** page of the Project window ([Figure 3.23](#)).

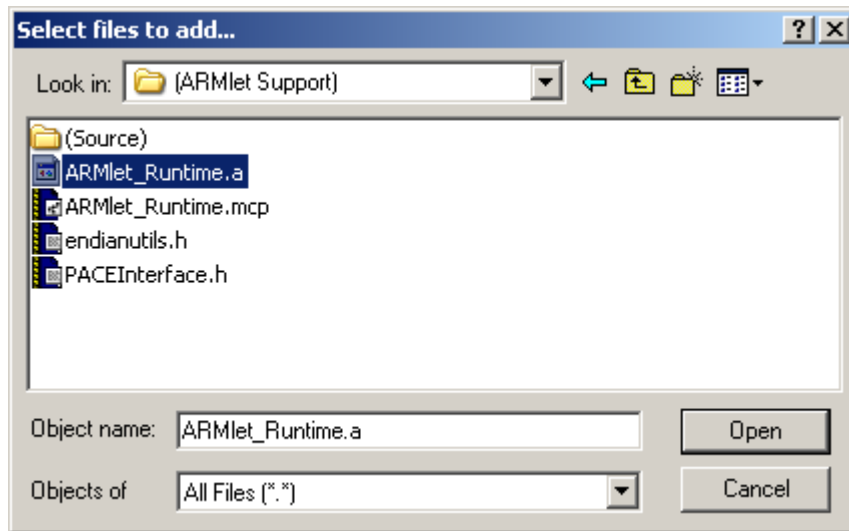
Figure 3.23 Project window - Files page



2. Select **Project > Add Files** from the CodeWarrior menu bar.

The IDE displays the **Select files to add** dialog box ([Figure 3.24](#)).

Figure 3.24 Select files to add dialog box

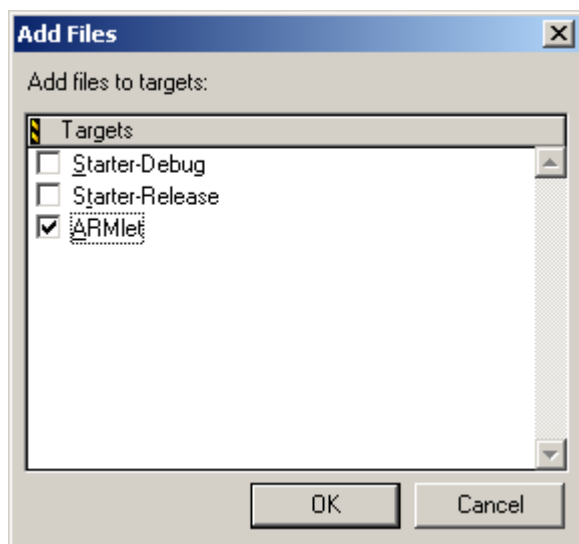


3. Navigate to the CW for Palm OS Support\ (ARMlet Support) folder within the CodeWarrior Development Tools for Palm OS installation folder and select the ARMlet_Runtime.a library file.

4. Click **Open**.

The IDE displays the **Add Files** dialog box ([Figure 3.25](#)). This dialog box contains checkboxes for all build targets in the project.

Figure 3.25 Add Files dialog box

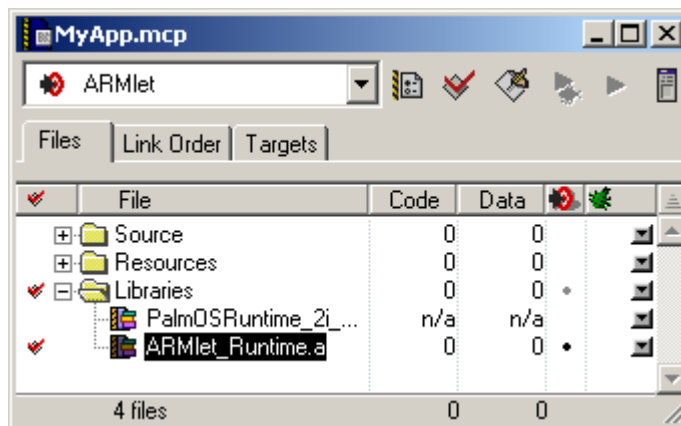


5. Uncheck the box corresponding to the application build target.
If there are multiple application build targets in the project, uncheck them all.
6. Check the box corresponding to the **ARMlet** build target.

7. Click **OK**.

The IDE adds the `ARMLet_Runtime.a` library file to the ARMLet build target and displays the library file in the Project window ([Figure 3.26](#)).

Figure 3.26 Project window - ARMLet runtime library added



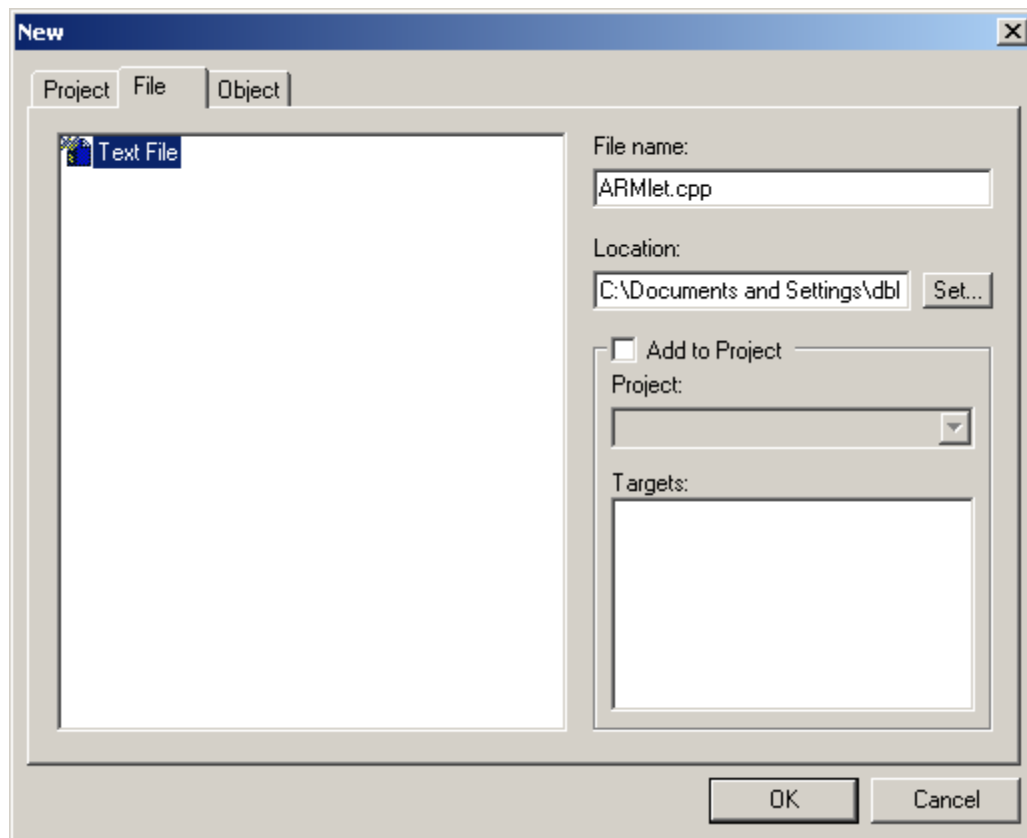
Write ARMlet Source Code

Next, write the ARMlet source code for the build target and write the code in the application that calls the ARMlet code resource.

1. Select **File > New** from the CodeWarrior menu bar.

The IDE displays the **New** window ([Figure 3.27](#)).

Figure 3.27 New window



2. Select **Text File**.
3. Enter a name for the ARMlet source code file in the **File name** text field, such as `ARMlet.cpp`.
4. Use the **Location** text field and **Set** button to set the location of the new file to the project folder for this project.

5. Click **OK**.

The IDE creates a new text file with the name and location you specified and displays the ARMlet source code editor window.

6. Write the ARMlet source code.

The ARMlet source code file should include the `PceNativeCall.h` header file and contain an entry function named `ARMlet_Main()`. Define the entry function to match the `NativeFuncType` type defined in `PceNativeCall.h`:

```
unsigned long ARMlet_Main(const void *emulStateP, void
*userData68KP, Call68KFuncType *call68KFuncP);
```

NOTE	Declare the <code>ARMlet_Main()</code> function as <code>extern "C"</code> if it is in a C++ source file. Otherwise, the linker will generate an error indicating that <code>ARMlet_Main()</code> is undefined.
-------------	---

7. Select **File > Save** from the CodeWarrior menu bar.

The IDE saves the ARMlet source code file.

8. Double-click the application source code file in the Project window.

The IDE opens the application source code file and displays it in an editor window.

9. Add code to the application source file that calls the ARMlet.

Add code to include the `PceNativeCall.h` header file, and use the `PceNativeCall()` function to call the ARMlet code from the application. The `PceNativeCall()` function takes two arguments: the address of the ARMlet, and a 32-bit value that Palm OS passes to the ARMlet as the `userData68KP` parameter.

The `userData68KP` parameter is a 32-bit value that the application passes to the ARMlet. The value may represent an integer, a float, a boolean, or anything else that fits into 32 bits (for example, a pointer to some data). ARMlets do not have direct access to the context of the calling 68K application; they cannot access the application's global variables or stack. Because of this, if an application developer wishes to pass data to an ARMlet, he or she must collect that data into a structure and pass a pointer to that structure to the ARMlet in the `userData68KP` parameter.

CAUTION The Palm OS automatically swaps the bytes of the `userData68KP` parameter value to represent the same value on the ARM processor. If the parameter value is a pointer, however, the ARMlet must manually swap any values within the structure to which it points. You should ensure that the ARMlet correctly handles byte alignment issues because it is possible for data allocated on the 68K side to not be correctly aligned on the ARM side.

10. Select **File > Save** from the CodeWarrior menu bar.

The IDE saves the application source code file.

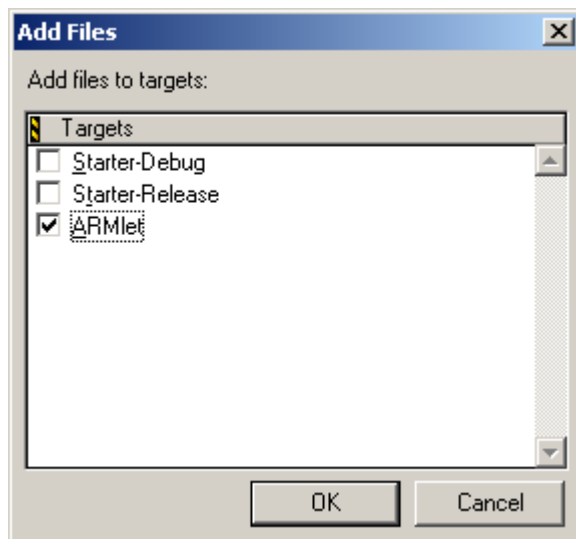
Add ARMlet Source Code to the Project

Next, add the ARMlet source code file you just created to the project.

1. Click inside the ARMlet source code editor window to activate the window.
The IDE brings forward the ARMlet source code editor window.
2. Select **Project > Add ARMlet.cpp to Project** from the CodeWarrior menu bar.

The IDE displays the **Add Files** dialog box ([Figure 3.28](#)). This dialog box contains checkboxes for all build targets in the project.

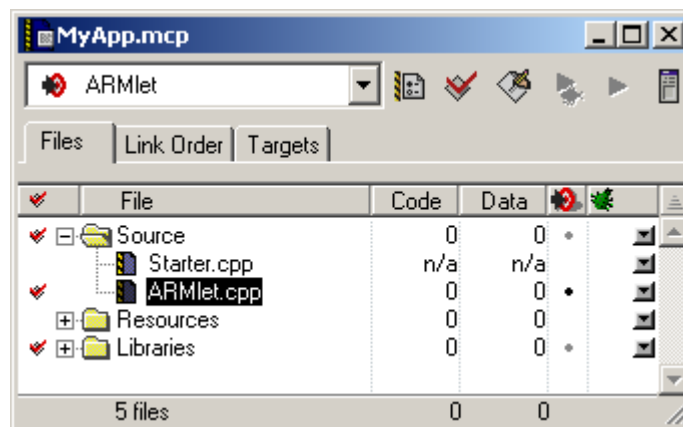
Figure 3.28 Add Files dialog box



3. Uncheck the box corresponding to the application build target.
If there are multiple application build targets in the project, uncheck them all.
4. Check the box corresponding to the **ARMlet** build target.
5. Click **OK**.

The IDE adds the ARMlet source code file to the **ARMlet** build target and displays the file in the Project window ([Figure 3.29](#)).

Figure 3.29 Project window - ARMlet source code file added



Create Build Target Dependencies

Next, link the ARMlet build target to the application build target so that the next time you build the application target, the IDE automatically builds the ARMlet target first.

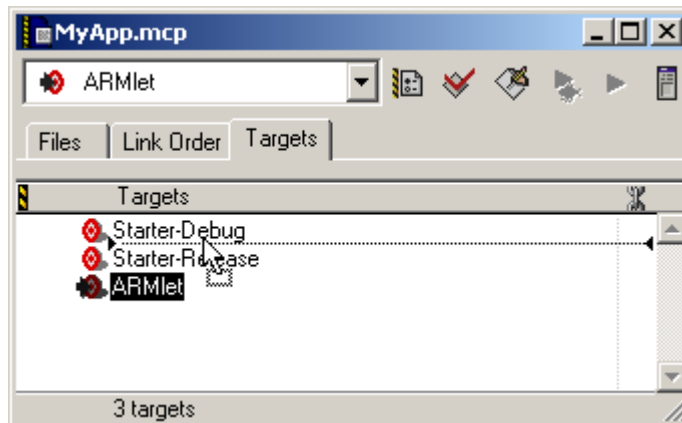
NOTE If the project contains more than one application build target, repeat the following procedure for each additional application build target.

1. Click the **Targets** tab in the Project window.

The IDE displays the **Targets** page of the Project window.

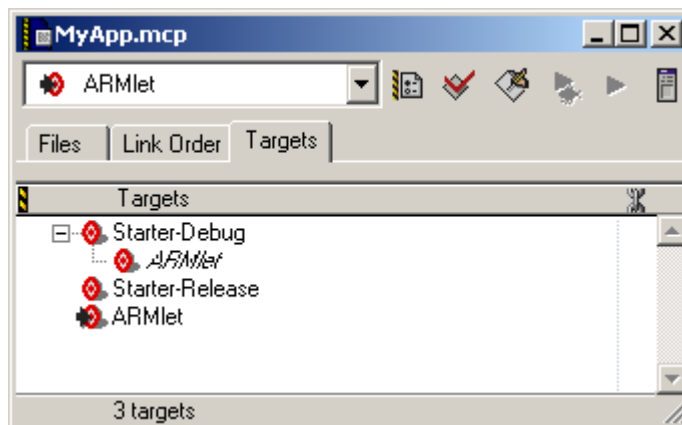
2. Drag the ARMlet build target icon and drop it over the application build target icon as shown in [Figure 3.30](#).

Figure 3.30 Project window - creating build target dependency



The IDE displays the ARMlet target name in italics under the application build target to indicate the dependency ([Figure 3.31](#)).

Figure 3.31 Project window - showing build target dependency



The IDE now considers the ARMlet build target a dependency to the application build target. As a result, the IDE will build the ARMlet target before building the application target the next time you build the application target.

Merge the ARMlet Executable Code Into the Application

In this section, you configure the project to merge the ARMlet executable code into the final application. The steps you must take depend on the linker you use to create the application:

- [Macintosh 68K Linker](#)
- [Palm OS 68K Linker](#)

Macintosh 68K Linker

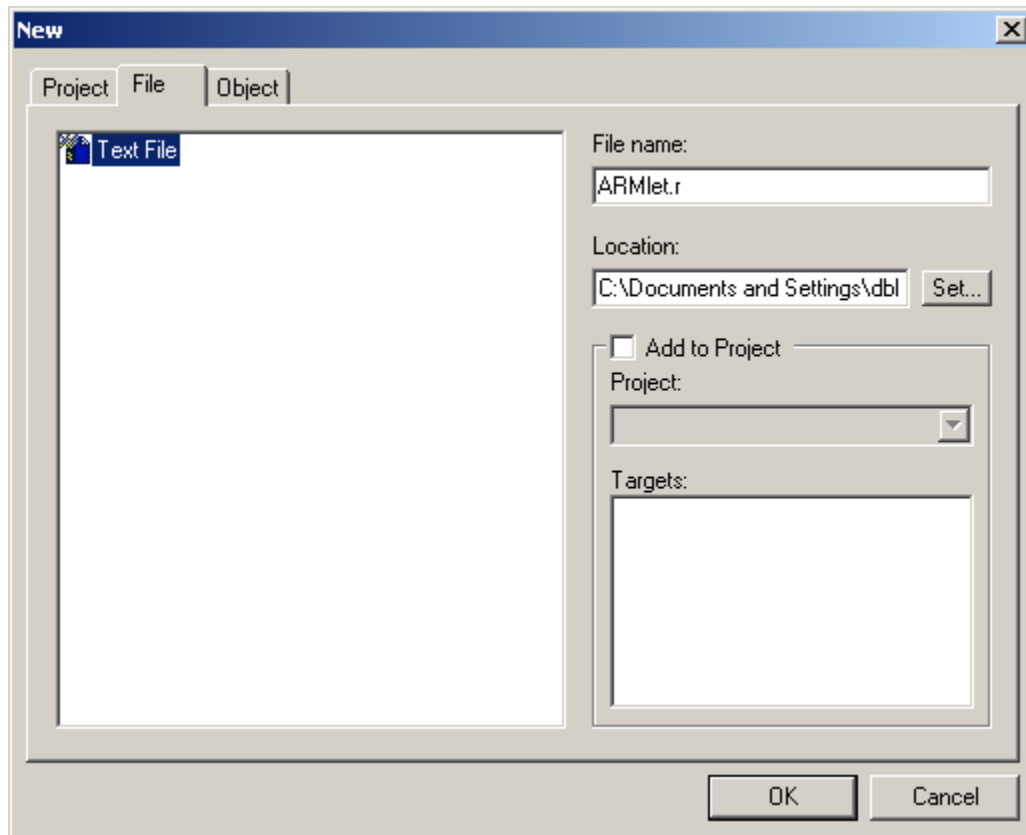
If the application to which you want to add the ARMlet uses the Macintosh 68K linker, the next step is to add a Rez source file to the application build target that instructs the Rez compiler to merge the ARMlet binary executable file into the application.

NOTE	If the application uses the Palm OS 68K linker, skip this section and proceed to “Palm OS 68K Linker” on page 85 .
-------------	--

1. Select **File > New** from the CodeWarrior menu bar.

The IDE displays the **New** window ([Figure 3.32](#)).

Figure 3.32 New window

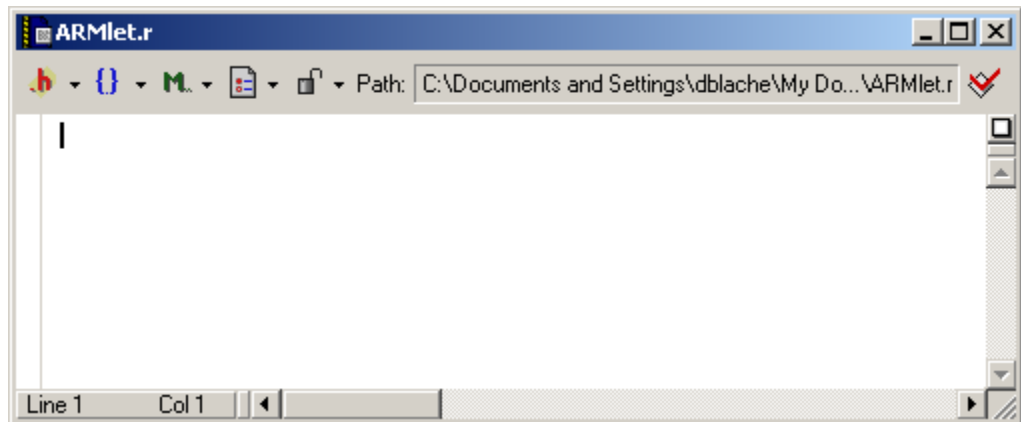


2. Click the **File** tab at the top of the window.
The IDE displays the **File** page of the New window.
3. Select **Text File**.
4. Enter a name for the ARMlet Rez source file in the **File name** text field, such as `ARMlet.r`.
5. Use the **Location** text field and **Set** button to set the location of the text file to the project folder.

6. Click **OK**.

The IDE closes the **New** window, creates the text file, and displays an editor window for the text file ([Figure 3.33](#)).

Figure 3.33 ARMlet.r editor window



7. Type this source code to the window (where *CCCC* is the four-character ARMlet creator code, *ID* is the ARMlet resource ID, and *Filename* is the ARMlet file name):

```
read 'CCCC' (ID) "Filename";
```

For example, if you specified a creator code of *ARMC*, a resource ID of 1000, and a filename of “*ARMC0001.bin*”, the Rez source code would be:

```
read 'ARMC' (1000) "ARMC03E8.bin";
```

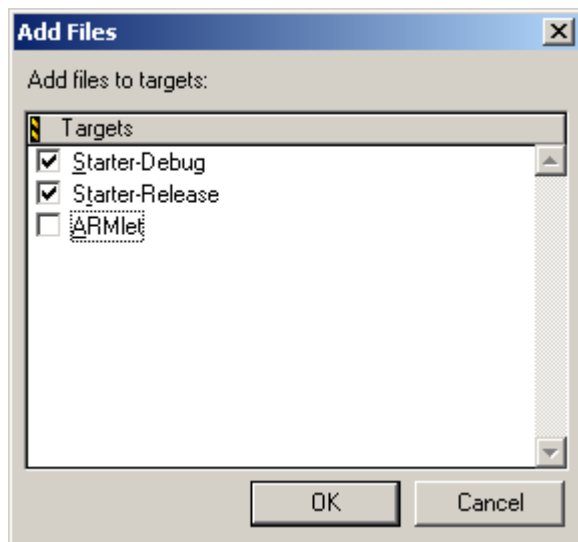
8. Select **File > Save** from the CodeWarrior menu bar.

The IDE saves your changes to the Rez source file.

9. Select **Project > Add Filename to Project** from the CodeWarrior menu bar (where *Filename* is the name of the Rez source file).

The IDE displays the **Add Files** dialog box ([Figure 3.34](#)).

Figure 3.34 Add Files dialog box



10. Check the box corresponding to the application build target.
If there are multiple application build targets in the project, check them all.
11. Uncheck the box corresponding to the **ARMlet** build target.
12. Click **OK**.

The IDE adds the Rez source file to the application build target.

The next time you build the target, the Rez compiler merges the ARMlet binary executable file into the application.

Palm OS 68K Linker

If the application to which you want to add the ARMlet uses the Palm OS 68K linker, the next step is to link the output of the ARMlet build target to the application build target. This causes the IDE to automatically merge the ARMlet build target output into the application's executable code when you build the application target.

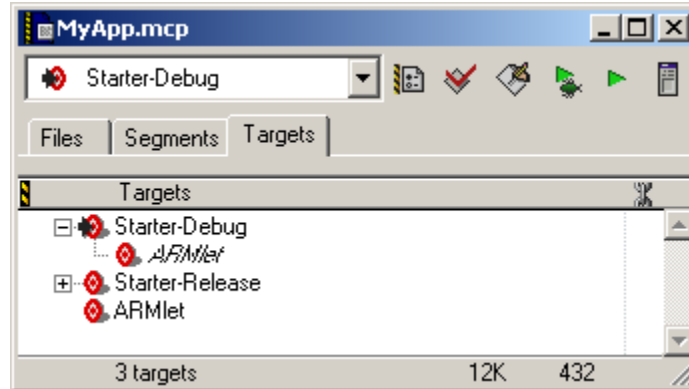
NOTE If the application uses the Macintosh 68K linker, skip this section and go back to [“Macintosh 68K Linker” on page 81](#).

NOTE If the project contains more than one application build target, repeat the following procedure for each additional application build target.

1. Click **Targets** in the Project window.

The IDE displays the **Targets** page of the Project window ([Figure 3.35](#)).

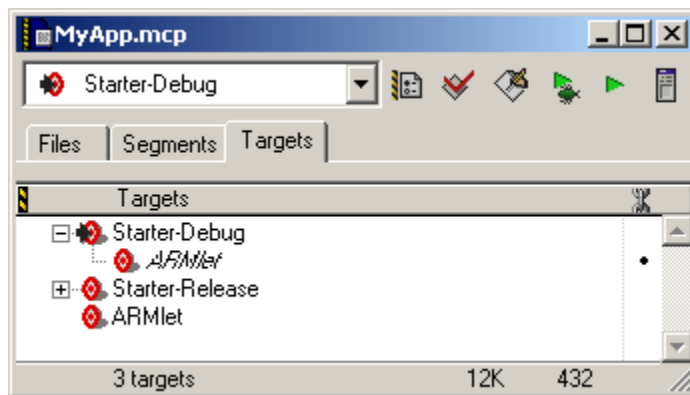
Figure 3.35 Project window - Targets page



2. Click the Link column area next to the italic ARMlet target name.

The IDE adds a mark in the column next to the dependency (as shown in [Figure 3.36](#)).

Figure 3.36 Project window - showing link against target setting



The IDE links the ARMlet build target output to the application build target. As a result, the IDE automatically includes the output of the ARMlet target, the ARMlet binary resource file, when you build the application target.

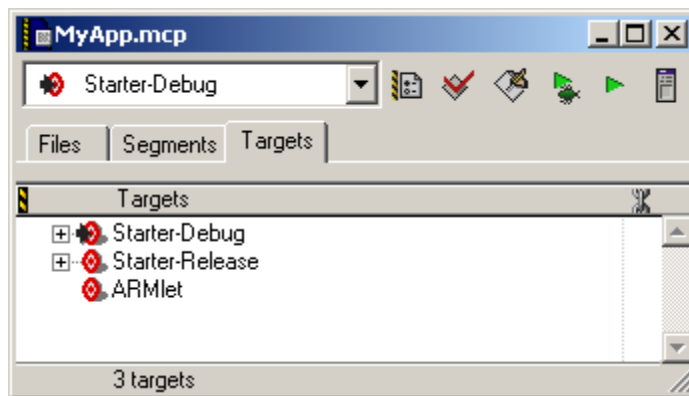
Build the Application Target

Finally, build the application target.

1. Select **Project > Set Default Target > *TargetName*** from the CodeWarrior menu bar (where *TargetName* is the name of the application build target).

The IDE switches to the application build target (as shown in [Figure 3.13](#)).

Figure 3.37 Project window - application build target selected



2. Select **Project > Make**.

The IDE builds the application target, combining the ARMlet code resource with the code in the application executable file. When you run the application on a ARM-based Palm OS device, the ARMlet executes within the application.

Congratulations! You have finished adding an ARMlet to your application!

Working With Applications

Adding ARMlets to Existing Projects

Working With Libraries

This chapter describes how to use the CodeWarrior IDE to create Palm OS® libraries, how to use Palm OS libraries in other CodeWarrior projects, and describes the CodeWarrior runtime and MSL libraries for Palm OS software development.

- [Types of Libraries](#) — describes the different types of Palm OS libraries you can create with the CodeWarrior IDE
- [Creating Static Library Projects](#) — shows how to create Palm OS libraries using the CodeWarrior static library stationery
- [Creating Shared Library Projects](#) — describes the CodeWarrior shared library wizard you use to create Palm OS shared library projects
- [Runtime Libraries](#) — describes the various runtime libraries needed to build Palm OS executables using the CodeWarrior IDE
- [MSL Libraries](#) — describes the Metrowerks Standard Libraries (MSL) that provide ANSI-standard library routines for C and C++ programming

Types of Libraries

The types of Palm OS libraries you can create using the CodeWarrior IDE are:

- [Static Libraries](#)
- [Shared Libraries](#)

Static Libraries

A *static library* is an executable file that contains functions that applications, code resources, and other libraries can use.

When you use a static library in a CodeWarrior project, the IDE links the static library with the other components of the project so that the static library functions become part of the executable code within the final executable file.

CAUTION	You must build static libraries with the same compiler settings as the applications that use them. For example, expanded mode settings and double / integer size settings should match. If you fail to match these settings you may encounter runtime problems such as mismatched program stacks and incorrect function parameter values.
----------------	---

If you create a static library that refers to other libraries, you do not need to add the other libraries to the project. Instead, you can declare the other library functions as external using the `extern` keyword so that CodeWarrior will link your static library without error.

NOTE	When you add your static library to an application project, you must also add to the project all external libraries to which your static library refers. To learn about which libraries to include in your project see “Runtime Libraries” on page 98 and “MSL Libraries” on page 101 .
-------------	---

Shared Libraries

Shared libraries are similar to static libraries (see [“Static Libraries” on page 90](#)) except that the library is a separate file from the application, code resource, or library that uses it. The Palm OS loads shared libraries into memory where multiple applications, code resources, or other shared libraries can use the shared library at the same time by using special system calls.

Developers use shared libraries to implement common functions across a set of applications and to reduce memory usage.

NOTE	Due to limited access to global variables, shared libraries have limited functionality.
-------------	---

For an example shared library project, see the examples in the “Palm OS 5.0 SDK Examples” folder located in the “(CodeWarrior Examples)” install folder within your CodeWarrior installation folder.

For the Palm OS whitepaper on shared libraries, see:

<http://oasis.palm.com/dev/kb/papers/1143.cfm>

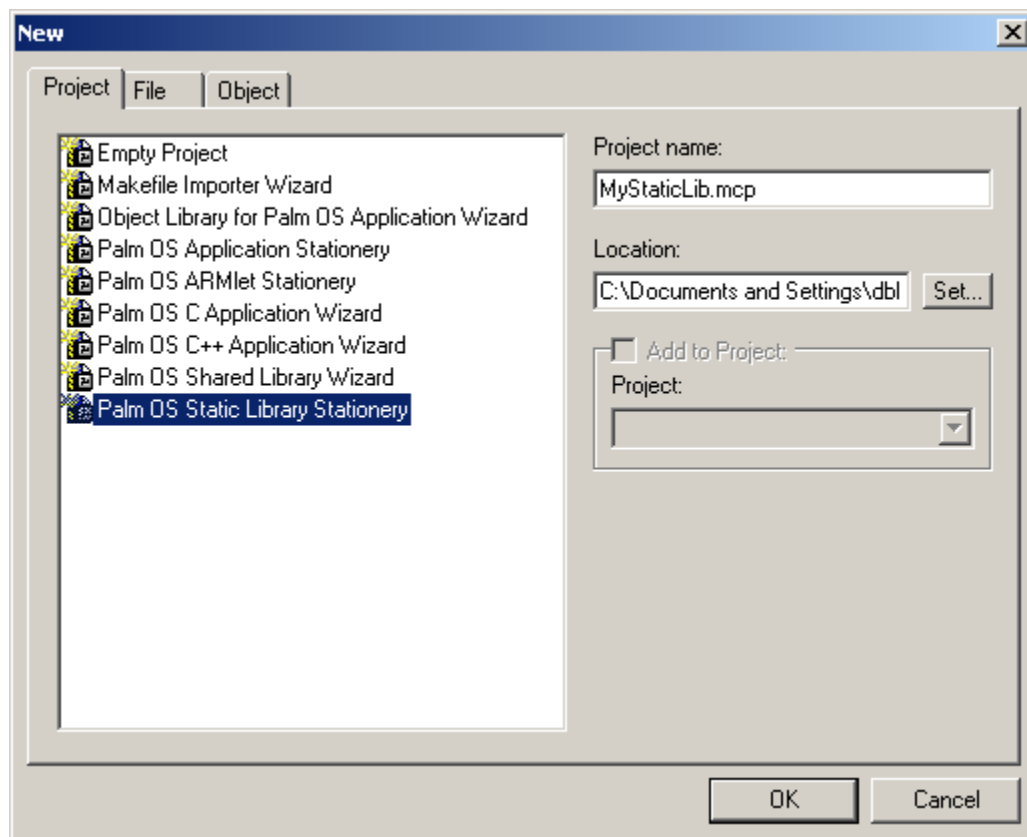
Creating Static Library Projects

This section shows you how to create static libraries for Palm OS using the CodeWarrior IDE. Creating a static library is very simple:

1. Run the CodeWarrior IDE
2. Choose **File > New** from the CodeWarrior menu bar.

The IDE displays the **New** window ([Figure 4.1](#)).

Figure 4.1 New window



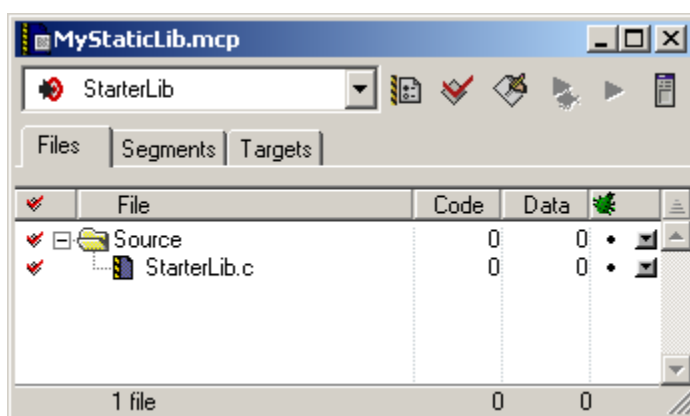
3. Select **Palm OS Static Library Stationery** from the list.
4. Enter a project name into the **Project Name** text field.

TIP The standard filename extension for CodeWarrior project files is .mcp.

5. Set the location for the project using the **Location** text field and the **Set** button.
6. Click the **OK** button.

The IDE creates the project and displays the Project window ([Figure 4.2](#)).

Figure 4.2 Project window



The StarterLib.c source file contains the source code for the library.

Creating Shared Library Projects

This section shows you how to create shared library projects using the CodeWarrior shared library wizard. The CodeWarrior shared library wizard interactively creates the project by using your input to generate projects with settings, libraries, source code files, resource files, and segmentation information tailored to your needs

To use the shared library wizard to create a Palm OS project:

1. Run the CodeWarrior IDE.
2. Select **File > New** from the CodeWarrior menu bar.
The IDE displays the **New** window.
3. Select the **Palm OS Shared Library Wizard**.
4. Enter a name for the project into the **Project name** text field.

TIP	The standard filename extension for CodeWarrior project files is .mcp.
------------	---

5. Click the **Set** button to set the location for the project folder.
6. Click **OK** to start the wizard.
7. Follow the instructions the wizard presents.

The rest of this section describes the shared library wizard included with the CodeWarrior Development Tools for Palm OS package.

Use the Palm OS Shared Library wizard to create Palm OS shared libraries written in the C language.

Page 1 of 3

The first page of the wizard ([Figure 4.3](#)) asks you for information about the shared library you want to create.

Figure 4.3 Palm OS Shared Library Wizard - Page 1 of 3

Palm OS Shared Library Wizard - Page 1 of 3

This wizard helps you create a new project to build a Palm OS shared library.

What is the name of the library?

MySharedLib

What is the name of the library's PRC file?

MySharedLib.prc

What is the creator ID of the library?

STRT Visit Creator ID Website

What prefix should be given to functions exported by this library?

MySharedLib

< Back Next > Cancel

Enter the name of the library. This is the name by which other developers identify the library.

Enter the name of the PRC file for the shared library. This is the name of the PRC file the CodeWarrior IDE generates when you build the project.

NOTE Developers sometimes use a shortened or abbreviated version of the shared library name for the PRC file name.

Enter the creator code for the shared library. The creator code should be a four-character code that is unique to your shared library. Click the **Visit Creator ID Website** button to view the Palm OS developer creator ID web site in your web browser.

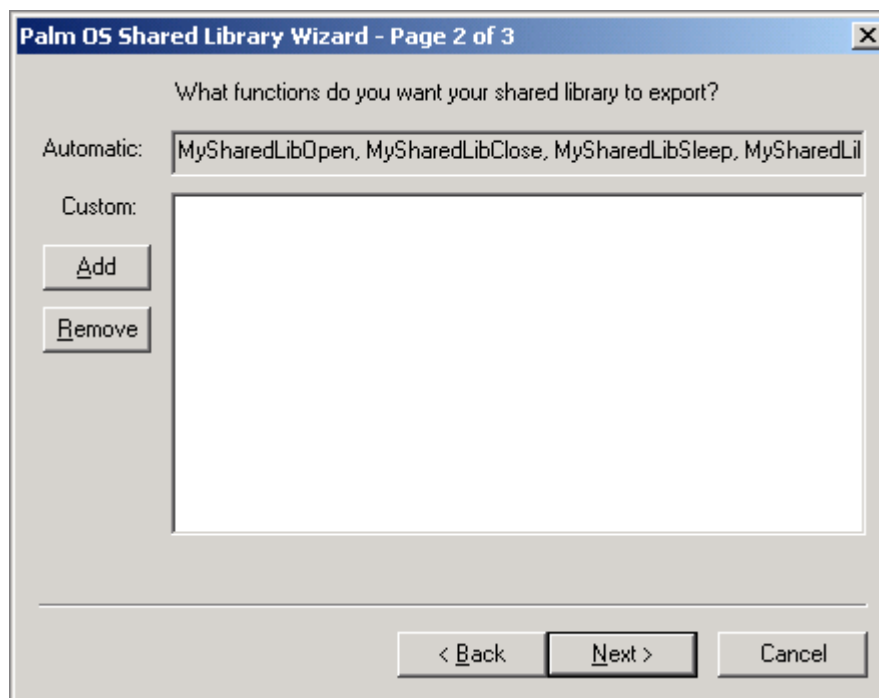
NOTE You should ensure that the creator ID you choose is unique and is registered with the Palm OS creator ID website before you make the application available to the public.

Enter the prefix you want library functions to have. The wizard prepends this text to the name of each function it generates in the shared library source code.

Page 2 of 3

The next page of the wizard ([Figure 4.4](#)) allows you to specify the functions your shared library makes available to applications. The wizard automatically generates prototypes and empty implementations for these functions.

Figure 4.4 Palm OS Shared Library Wizard - Page 2 of 3



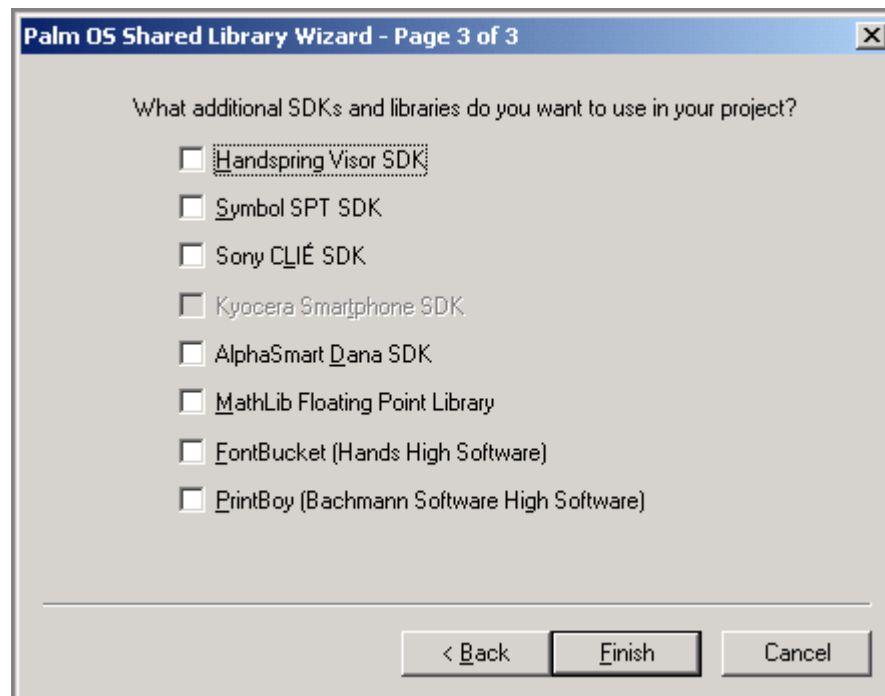
The **Automatic** text field lists the functions that the wizard automatically creates for you.

The **Custom** list shows the additional function names you want the wizard to generate for the shared library. To add another function to the Custom list, click the **Add** button and type the name of the function. Select a function in the **Custom** list and click **Remove** to remove a function from the list.

Page 3 of 3

On this page ([Figure 4.5](#)), the wizard asks you to choose the software development kits (SDKs) and libraries you want to use in the project.

Figure 4.5 Palm OS Shared Library Wizard - Page 3 of 3



Check the boxes next to the SDKs and libraries you want to use in your application.

NOTE	The list of SDKs you see in this page of the wizard may be different in your particular installation of CodeWarrior Development Tools for Palm OS.
-------------	--

Runtime Libraries

This section provides a reference to the CodeWarrior runtime libraries required to create different types of Palm OS executables using the CodeWarrior IDE, and describes issues relating to the usage of those runtime libraries in your projects.

- [Runtime Library Reference](#)
- [Multi-Segment Project Libraries](#)

Runtime Library Reference

This section lists the runtime libraries you need to build different types of CodeWarrior projects.

[Table 4.1](#) lists the libraries you need for each type of Palm OS project. When you use project stationery or wizards to create a new project, the IDE automatically adds the required runtime libraries to the new project for you. You can use this table to verify that you have the correct runtime libraries in your projects or as a guide to creating projects from scratch.

Table 4.1 Runtime libraries for Palm OS application projects

Type of Project	Required Runtime Libraries
single-segment C application	PalmOSRuntime_2i_A5.lib
single-segment C++ application (2 byte integers)	PalmOSRuntime_2i_A5.lib
single-segment C++ application (4 byte integers)	PalmOSRuntime_4i_A5.lib
multi-segment application (2 byte integers)	PalmOSRuntime_2i_A5.lib
multi-segment application (4 byte integers)	PalmOSRuntime_4i_A5.lib
expanded mode application (2 byte integers)	PalmOSRuntime_2i_A4A5.lib
expanded mode application (4 byte integers)	PalmOSRuntime_4i_A4A5.lib

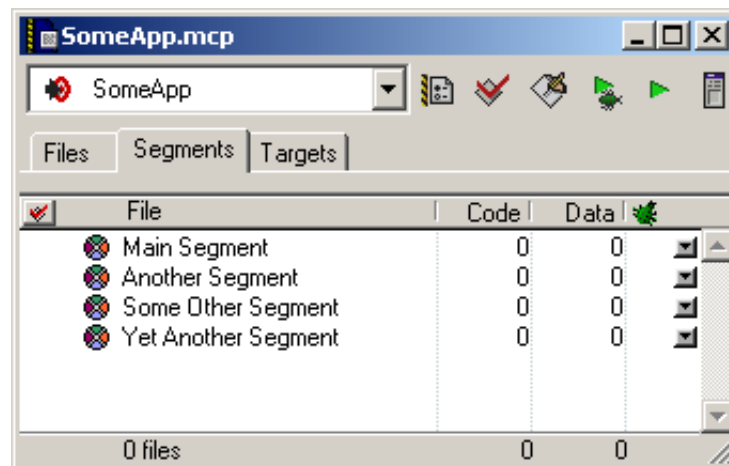
The option to use 4 byte integers, or the default 2 byte integers, is located in the **68K Processor** settings panel. See [“68K Processor” on page 159](#) for more information on this settings panel.

Multi-Segment Project Libraries

This section describes how to correctly arrange runtime libraries in multi-segment projects. Multi-segment applications consist of multiple segments of code that the operating system loads into memory as needed.

You can specify the code the IDE places into each segment of your application using the **Segments** page of the Project window (shown in [Figure 4.6](#)).

Figure 4.6 The Segments page of the Project window



To access the Segments page, click the **Segments** tab in the Project window. The Segments page lists all of the segments in the project. You arrange segments in this page by dragging them to different positions in the list.

The linker creates the first segment, segment 0, which contains instructions and data the operating system uses at runtime for loading and preparing other segments in the application for execution. This segment does not appear in the **Segments** page of the Project window.

The next segment, segment 1, contains startup code and application code. This segment must contain the following items:

- one of the runtime libraries listed in [Table 4.1 on page 98](#)
- the source code file containing the `PilotMain()` function
- the source code files containing any other routines that the `PilotMain()` function calls when it receives a launch command that is not `sysAppLaunchCmdNormalLaunch`.

Subsequent segments also contain application code. You typically organize these segments to perform a related set of features. Good candidates for a code segment include a form's handling routines and static routines.

NOTE	Reduce the number of inter-segment function calls whenever possible. Function calls within a segment execute faster than function calls between segments. In addition, each inter-segment call reduces the available global data space; every function you call from a different segment requires an additional entry in the global jump table.
-------------	---

MSL Libraries

The Metrowerks Standard Libraries (MSL) provide ANSI-standard library routines for C and C++ programming:

- [Standard C Library](#)
- [Standard C++ Library](#)

The easiest way to support MSL in your Palm OS applications is to create a new application using the Palm OS application wizard and to select the MSL option.

To add support for MSL to an existing build target, use the **File Mappings** target settings panel to add a system access path that points to {Compiler}CW for Palm OS Support\MSL. For instructions about how to convert older projects to CodeWarrior Development Tools for Palm OS version 9, see [“Converting Older Projects to Version 9” on page 40](#).

NOTE	For more detailed information about MSL, refer to the <i>MSL C Reference</i> and <i>MSL C++ Reference</i> manuals.
-------------	--

Standard C Library

This implementation of MSL C provides a full set of standard headers for a freestanding implementation as defined by section 4 of the ISO C (1999) document:

- `<float.h>`
- `<iso646.h>`
- `<limits.h>`
- `<stdarg.h>`
- `<stdbool.h>`
- `<stddef.h>`
- `<stdint.h>`

In addition to these headers, this MSL C implementation also provides the standard library functions:

<stdlib.h>

Table 4.2 **stdlib.h** functions

abort	exit	calloc
malloc	free	realloc
atol	atoi	RAND_MAX
div_t	ldiv_t	abs
labs	div	ldiv
srand	rand	

<string.h>

Table 4.3 **string.h** functions

memcpy	memmove	memcmp
memchr	strlen	strcpy
strcmp	strstr	strcat
strncat	strncpy	

<time.h>

- CLOCKS_PER_SEC
- clock_t, clock

<stdio.h>

- EOF

Standard C++ Library

This implementation of MSL C++ supports standard template libraries, but lacks support for locales, numerics, and file/console input and output.

The Palm OS MSL C++ implementation fully supports the standard headers listed in [Table 4.4](#).

Table 4.4 MSL C++ supported standard headers

algorithm	bitset	deque
exception	functional	iosfwd
iterator	limits	list
map	memory	new
numeric	queue	set
stack	stdexcept	string
typeinfo	utility	vector

The headers listed in [Table 4.5](#) do not have any functional file I/O or console I/O

Table 4.5 Headers without functional I/O or console I/O

iomanip	ios	istream
ostream	sstream	streambuf
stringstream		

The following standard headers are not available for Palm OS:

Table 4.6 Headers not available for Palm OS

fstream	complex
iostream	valarray
locale	

Working With Resources

This chapter discusses how to create resources for use in your Palm OS® projects, how to incorporate them into your projects, and how to convert Macintosh resources to use with the latest tools for Palm OS software development.

- [Resource Guidelines](#) — provides a description of Palm OS resources and guidelines for creating and working with resources
- [Creating Palm OS® Resources](#) — shows how to create Palm OS resources using the CodeWarrior IDE
- [Adding Resource Files To Projects](#) — describes how to add Palm OS resources to existing CodeWarrior projects
- [Converting Resource Files](#) — describes how to convert Macintosh resource files to resource compiler source (RCP) files

Resource Guidelines

Palm OS resources are additional data items, separate from the executable code, that an application uses to hold descriptions of user interface items like forms, alerts, and text. Because resources are separate from the executable code, it is possible to modify the behavior and appearance of an application without having to modify its executable code.

We refer to Palm OS resources by their resource type and resource ID. Every Palm OS resource has a case-sensitive four-character type code and a unique, 16-bit signed integer ID. For example, a button resource has a 'tBTN' type. Palm OS IDs can range from 0 to 32678, but typically follow these numbering conventions:

- ID values greater than 10000 are reserved for system resources. Applications must not have ID values greater than 10000.
- Form resource ('tFRM') IDs begin at 1000 and are incremented in steps of 100. For example, a database application has two forms: a form that lists records and another form contains a single record's fields. The list form has an ID of 1000. The single record form has an ID of 1100.
- User interface resources within a form, such as button, check box, and field resources, have IDs based on the form's ID. For example, the database application's list form might have a list resource ('tLST') with an ID of 1001.
- Even though some resources of different types (such as a Form and a Menu) may share the same ID, it is better to have unique IDs to avoid confusion. Resources on the same form (such as a button and a list) may not share the same ID.
- Bitmaps and bitmap families must all have unique IDs. IDs of these types are not allowed to overlap. The same goes for icon and icon families.
- If a resource has an ID field, make sure its ID value matches the ResEdit ID value.

NOTE	For the most part, Constructor for Palm OS follows these guidelines automatically by correctly numbering resources for you as you create them in a Constructor project. The only exceptions are bitmaps, icons, and their families.
-------------	---

Creating Palm OS® Resources

This section describes various utilities you can use to create resources to use with your Palm OS projects:

- [PilRC Designer](#)
- [Constructor for Palm OS](#)
- [Other Utilities](#)

PilRC Designer

PilRC Designer is a visual resource editing tool that is installed with CodeWarrior Development Tools for Palm OS version 9. You use PilRC Designer to create the user interfaces for your Palm OS projects. It provides a WYSIWYG (What You See Is What You Get) view of a Palm Pilot organizer's screen onto which you lay out forms, alerts, and other interface objects such as bitmaps.

This is the preferred resource editor for projects that use the new build tools (see [“The New Build Tools” on page 17](#)) because it outputs resources in RCP files that you can include in your CodeWarrior projects.

To run PilRC Designer, select **PalmOS > Launch PilRC Designer** from the CodeWarrior menu bar.

PilRC Designer is located in the following folder (where *CWFolder* is the folder where you installed the CodeWarrior Development Tools for Palm OS package):

`CWFolder\CW For Palm OS Tools\PilRC Designer\`

Constructor for Palm OS

Constructor for Palm OS is a visual resource editing tool that allows you to create user interfaces for your Palm OS projects. Like PilRC Designer, Constructor for Palm OS provides a WYSIWYG view of a Palm Pilot connected organizer's screen onto which you lay out forms, alerts, and other interface objects.

Unlike PilRC Designer, Constructor for Palm OS does not output RCP files - instead, it outputs Macintosh resource (.rsrc) files. Because it outputs Macintosh resource files, Constructor for Palm OS is best used with projects that use the old build tools (see [“The Old Build Tools” on page 14](#)).

Select **PalmOS > Launch Constructor for Palm OS** from the CodeWarrior menu bar to run Constructor for Palm OS.

Other Utilities

You can choose not to use the CodeWarrior resource editing utilities and instead specify resources using text files.

You can generate resources from Macintosh-based Rez source (.r) files. Rez source files are text files written in the Rez resource description language. If you use Rez source files, refer to the Palm OS SDK file `UIResDefs.r` for definitions of Palm OS resource types.

You can also use the PilRC compiler (see [“PilRC Compiler” on page 22](#)) to process resource compiler source (RCP) files. We recommend you use RCP files rather than Rez source files because RCP files integrate more smoothly into the new build process (see [“The CodeWarrior Build Process” on page 14](#)). If you use RCP files, refer to the following document for information on the RCP file format (where *CWFolder* is the folder where you installed the CodeWarrior Development Tools for Palm OS package):

`CWFolder\CW for Palm OS Tools\PilRC\doc>manual.htm`

Adding Resource Files To Projects

In order to add resources to a project that creates an application that will run on a Palm OS device, select **Project > Add Files** from the CodeWarrior menu bar.

In order to run under the [Palm OS Simulator](#), add the resource files to the project, then create a resource list file. The resource list file is a C source code file that lists each resource file's path and name as character strings in a character string array named `AppResourceList`. The last entry in the `AppResourceList` must be an empty string (`" "`).

This C source file must also be added to your project. If your project has a target specifically for [Palm OS Simulator](#), add the C source code file to this target. Do not add the source code to any targets that generate an application for a Palm OS device.

[Listing 5.1](#) is an example `Rsc.c` file.

Listing 5.1 Example resource list file

```
/*
 * File:      SockSorterRsc.c
 * Purpose:   List resources used in SockSorter database application
 *           for Palm OS to run on Palm OS Simulator.
 */

#include <Pilot.h>

#if      LANGUAGE == LANGUAGE_ENGLISH
#define RESOURCE_FILE_PREFIX ""
#elif    LANGUAGE == LANGUAGE_FRENCH
#define RESOURCE_FILE_PREFIX "French:"
#elif    LANGUAGE == LANGUAGE_GERMAN
#define RESOURCE_FILE_PREFIX "German:"
#endif

// Application resources.

char *AppResourceList[] = {
    ":Rsc:SockSorterMain.rsrc",      // main form
    ":Rsc:SockSorterInfo.rsrc",     // record form
    ""                               // last entry
};
```

Converting Resource Files

This section describes various ways to convert Macintosh resource files to resource compiler source (RCP) files that the PilRC compiler merges into the final PRC file.

- [History of Palm OS Resource Compilers](#)
- [Using Conversion Utilities](#)
- [Using the Mac OS Merge Linker and PalmRez Post-Linker](#)

History of Palm OS Resource Compilers

Because initial versions of Palm OS development tools were based on Macintosh development tools, Palm OS developers used Apple Computer's Rez compiler and a template header file Palm OS designers created to generate resources. Metrowerks then created Constructor for Palm OS, which became the leading visual resource editor for Palm OS development.

In 1997, Wes Cherry created the Pilot Resource Compiler (PilRC), a third-party resource compiler for Palm OS. PilRC is similar to the Windows Resource Compiler (WinRC). They have similar language structures and even have similar source filename extensions (WinRC uses ".rc", PilRC uses ".rcp"). PilRC frees Palm OS developers from their dependency on Macintosh resource files by allowing them to generate resources by writing resource source (RCP) files.

Neil Rhodes did the Palm OS community a service by adapting PilRC to work as a CodeWarrior IDE plug-in module. The plug-in module translated RCP files into Rez compiler source (.r) files. This tool was an optional part of CodeWarrior Development Tools for Palm OS version 7 and was a standard part of CodeWarrior Development Tools for Palm OS version 8.

Starting with version 9 of the CodeWarrior Development Tools for Palm OS tools, the IDE has a built-in command-line adapter for the PilRC command-line compiler. This allows developers to utilize the full capabilities of the PilRC. It also speeds up the build process significantly.

Using Conversion Utilities

There are several resource conversion utilities available that allow you to convert Macintosh resource files to RCP files:

- [PilRC Designer](#)
- [rsrc2rcp \(PalmSource\)](#)
- [rsrc2rcp \(Open Source\)](#)
- [PRCExplorer](#)

PilRC Designer

PilRC Designer is a visual resource editing tool that is part of CodeWarrior Development Tools for Palm OS version 9. Although this tool does much more, you can use this tool to open Macintosh resource files and save them as RCP files. This tool supports high-resolution bitmap forms.

To run PilRC Designer, select **PalmOS > Launch PilRC Designer** from the CodeWarrior menu bar.

rsrc2rcp (PalmSource)

The `rsrc2rcp` program, written by Renaud Malaval, a developer at PalmSource in France, is a freeware utility that converts Macintosh resource files to RCP files by analyzing a Macintosh resource file and an associated header (.h) file to generate labelled items in the resulting RCP file. This program supports recent PilRC features to allow high-density bitmaps.

For more information, and to download this utility, go to:

<http://www.palmgear.com/software/showsoftware.cfm?prodID=42412>

rsrc2rcp (Open Source)

The open source version of `rsrc2rcp` is a Perl script that you can modify to suit your needs that converts Macintosh resource files to RCP files by analyzing a Macintosh resource file and an associated header (`.h`) file to generate labelled items in the resulting RCP file.

NOTE	At the time of this writing, the bitmap support in this utility is not as good as the PalmSource <code>rsrc2rcp</code> program.
-------------	---

For more information, and to download this utility, go to:

<http://rsrc2rcp.sourceforge.net/>

PRCExplorer

PRCExplorer is similar to a disassembler in that it extracts Palm OS resources from an existing PRC file, outputting them in an RCP file. This utility supports high-density bitmap forms.

For more information about this utility, go to:

<http://www.palmgear.com/software/showsoftware.cfm?prodID=40542>

Using the Mac OS Merge Linker and PalmRez Post-Linker

This section shows you how to set up a CodeWarrior project to use the Mac OS Merge linker (see [“Mac OS Merge Linker” on page 25](#)) and PalmRez post-linker (see [“PalmRez Post-Linker” on page 25](#)) to convert Macintosh resource files in a project to intermediate resource object (RO) files that the [Palm OS 68K Linker](#) later adds to the final Palm OS resource collection (PRC) file.

This procedure contains numerous steps that you must follow sequentially:

1. [Open the Project](#)
2. [Create a New Build Target](#)
3. [Configure the New Build Target](#)
4. [Add the Resource File to the New Build Target](#)
5. [Link the New Build Target to the Main Build Target](#)
6. [Build the New Target](#)
7. [Add the Resource Object File to the Project](#)
8. [Build the Main Target](#)

Open the Project

First, open the project that has the Macintosh resource file you want to convert. This project must use the [Palm OS 68K Linker](#) described in [“The New Build Tools” on page 17](#).

1. Select **File > Open** from the CodeWarrior menu bar.
The IDE displays the **Open** dialog box.
2. Navigate to the folder containing the project file you want to open.
3. Click **Open**.

The IDE opens the project and displays the Project window.

Create a New Build Target

Next, create a new build target. You will use this build target to convert the Macintosh resource file when you build the project.

1. Click the **Targets** tab in the Project window.

The IDE displays the **Targets** page of the Project window.

2. Select **Project > Create Target** from the CodeWarrior menu bar.

The IDE displays the **New Target** window.

3. Enter “Convert Resources” in the **Name for new target** text field.

4. Select the **Empty target** option.

5. Click **OK**.

The IDE closes the **New Target** window and creates the new build target. The IDE displays the new target in the **Targets** page of the Project window.

Configure the New Build Target

Next, configure the new build target to use the Mac OS Merge linker and PalmRez post-linker to convert the Macintosh resource file.

1. Select **Project > Set Default Target > Convert Resources** from the CodeWarrior menu to make “Convert Resources” the current build target.

The IDE switches to the “Convert Resources” build target. The IDE displays an arrow in the icon of the build target in the **Targets** page of the Project window to indicate it is the current build target.

2. Select **Edit > Convert Resources Settings** from the CodeWarrior menu bar.

The IDE displays the **Target Settings** window.

3. Click **Target Settings** in the **Target Settings Panels** list on the left side of the Target Settings window.

The IDE displays the **Target Settings** panel.

4. Select **Mac OS Merge** from the **Linker** menu.

5. Select **PalmRez Post-Linker** from the **Post-Linker** menu.
6. Click **Mac OS Merge** in the **Target Settings Panels** list on the left side of the Target Settings window.

The IDE displays the **Mac OS Merge** settings panel.
7. Select **Resource File** from the **Project Type** menu.
8. Enter “resources.tmp” in the **File Name** text field.

The Mac OS Merge linker outputs the resources into a file with this name when you build the project.
9. Uncheck the **Copy Code Fragments** box.
10. Click **PalmRez Post-Linker** in the **Target Settings Panels** list on the left side of the Target Settings window.

The IDE displays the **PalmRez Post-Linker** settings panel.
11. Enter “resources.tmp” into the **Mac Resource Files** text field.
12. Enter “resources.ro” into the **Output File** text field.
13. Enter “RSRC” into the **Type** field.
14. Uncheck the **Set Backup Bit** box.
15. Uncheck the **Sort Resources by Size** box.
16. Click **OK**.


The IDE saves your changes and closes the **Target Settings** window.

Add the Resource File to the New Build Target

Next, add the Macintosh resource file to the new build target. You assign the Macintosh resource file to the new build target so that the Mac OS Merge linker will process it when you build the project.

1. Click the **Files** tab in the Project window.

The IDE displays the **Files** page of the Project window.

2. Click the Target column () next to the Macintosh resource file to add a mark next to the file.

The IDE includes the Macintosh resource file in the “Convert Resources” build target.

Link the New Build Target to the Main Build Target

Next, link the new build target to the main build target so that the IDE will always automatically build the new target before building the main target.

1. Click the **Targets** tab in the Project window.

The IDE displays the **Targets** page of the Project window.

2. Drag the **Convert Resources** icon and drop it on top of the main build target for the project.

The IDE links the **Convert Resources** build target to the main build target. This causes the IDE to always build the Convert Resources target before building the main build target.

Build the New Target

Next, build the new target. You must do this now to have the Mac OS Merge linker and PalmRez post-linker generate the intermediate `resources.ro` file so that you can add the file to the project in the next section.

1. Select **Project > Set Default Target > Convert Resources** from the CodeWarrior menu to make “Convert Resources” the current build target.

The IDE switches to the “Convert Resources” build target.

2. Select **Project > Make** from the CodeWarrior menu bar.

The IDE builds the **Convert Resources** target. When the build is finished, you should see two new files in the project folder: `resources.tmp`, and `resources.ro`.

Add the Resource Object File to the Project

Next, add the `resources.ro` file that the Mac OS Merge linker and PalmRez post-linker generated in the previous section to main build target in the project. Once you add this file to the main build target, the [Palm OS 68K Linker](#) will include the resources in the final PRC file the next time you build the main target.

1. Click the **Files** tab in the Project window.

The IDE displays the **Files** page of the Project window.

2. Click the **Resources** group in the Project window to select it.

The IDE highlights the **Resources** group in the project window.

3. Select **Project > Add Files** from the CodeWarrior menu bar.

4. The IDE displays the **Select files to add** dialog box.

5. Navigate to the project folder and select the `resources.ro` file.

6. Click **Open**.

7. The IDE displays the **Add Files** dialog box.

This dialog box is where you specify the build targets to which you want the IDE to add the file.

8. Uncheck all build targets except the main build target.
9. Click **OK**.

The IDE adds the `resources.ro` file to the main build target. The IDE displays the file in the **Resources** group in the Project window,

Build the Main Target

Finally, build the main target. The IDE will generate the final PRC file, including the resources from the `resources.ro` file.

1. Select **Project > Set Default Target > *MainTargetName*** (where *MainTargetName* is the name of the main build target in the project).

The IDE switches to the main build target.

2. Select **Project > Make**.

The IDE builds the main target, including the resources from the `resources.ro` file.

At this point, the Macintosh resources are included in the final PRC file.

Target Settings Reference

This chapter describes the CodeWarrior IDE settings panels that affect code generation for Palm OS® platform development. By modifying the settings for the individual items within a panel you can control the compiler, linker, and other aspects of code generation.

NOTE	For specific details about the CodeWarrior compilers and linkers for Palm OS development, such as compiler pragmas, linker symbols, and so forth, see “C and C++ Compilers” on page 177 .
-------------	---

This chapter has these sections:

- [Other Target Settings Documentation](#)
- [Target Settings Window Overview](#)
- [Target Settings](#)
- [68K Target](#)
- [Palm OS 68K Target](#)
- [Palm OS ARMlet Target](#)
- [PRC File Settings](#)
- [PiIRC Settings](#)
- [Rez](#)
- [68K Processor](#)
- [68K Disassembler](#)
- [68K Linker](#)
- [ELF Disassembler](#)
- [PalmRez Post-Linker](#)
- [Palm OS Debugging](#)

Other Target Settings Documentation

This manual documents only those settings panels of specific interest to Palm OS developers. [Table 6.1](#) shows where to find documentation on the settings panels not documented in this manual.

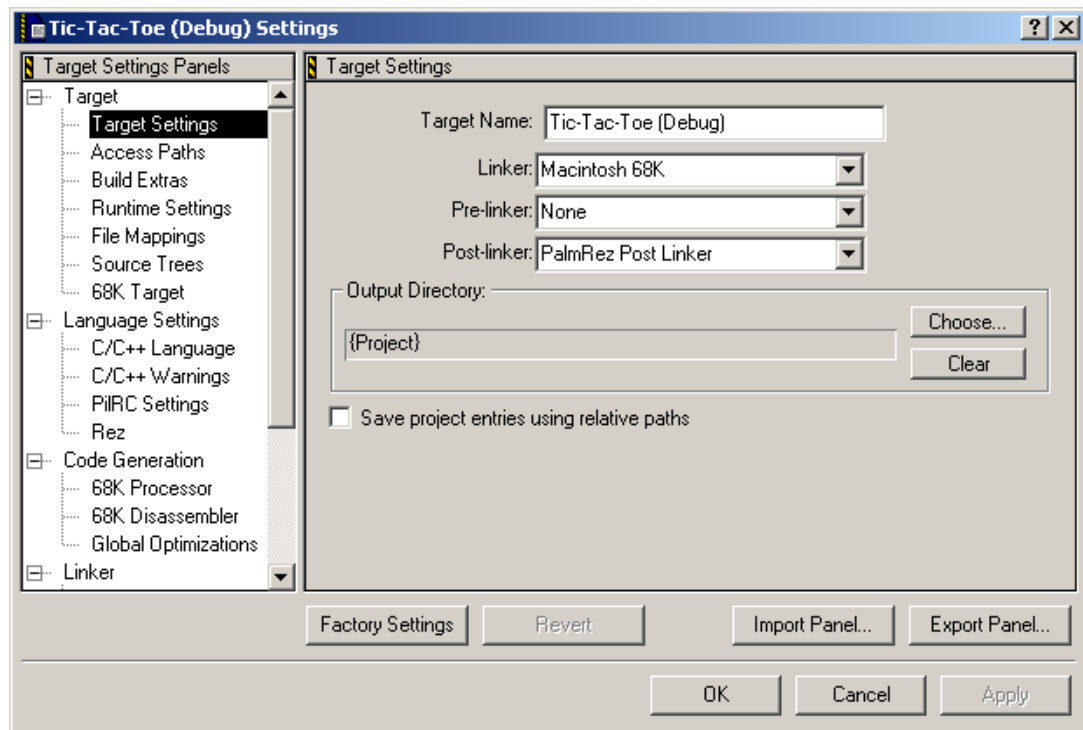
Table 6.1 Other target settings panel documentation

Settings Panel	Manual
Access Paths	<i>IDE User's Guide</i>
Build Extras	
Runtime Settings	
File Mappings	
Source Trees	
Global Optimizations	
Custom Keywords	
Other Executables	
Debugger Settings	
C/C++ Language C/C++ Warnings	<i>C and C++ Compilers Reference</i>

Target Settings Window Overview

A CodeWarrior project may contain multiple build targets. Each build target in a project has its own settings, called *target settings*. These settings control a variety of features such as compiler options, linker output, error and warning messages, and so forth. You modify these settings in the **Target Settings** window ([Figure 6.1](#)).

Figure 6.1 Target Settings window

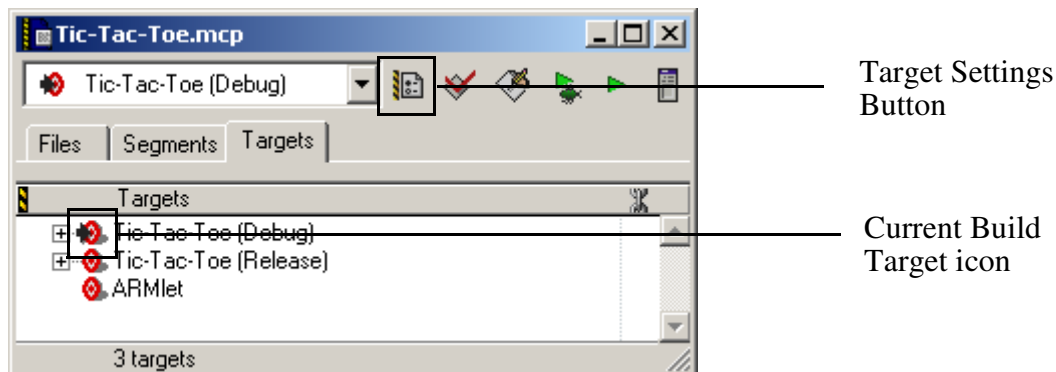


There are three ways to view the **Target Settings** window:

- Select **Edit > TargetName Settings**, (where *TargetName* is the name of the current build target).
- Double-click the icon of the build target in the **Targets** page of the Project window.
- To edit the current build target's settings, click the Target Settings button in the Project window ([Figure 6.2](#)).

NOTE The Current Build Target icon (🔧) in the **Targets** page of the Project window indicates the current build target (as shown in [Figure 6.2](#)).

Figure 6.2 Target Settings button in the Project window



TIP You can view multiple **Target Settings** windows to edit the settings for two or more build targets simultaneously.

To modify a build target's settings:

1. Select **Project > Set Default Target > *TargetName*** from the CodeWarrior menu bar (where *TargetName* is the name of the build target you want to configure).

The IDE switches to the build target you specified.

2. Select **Edit > *TargetName* Settings** from the CodeWarrior menu bar (where *TargetName* is the name of the current build target).

The IDE displays the Target Settings window.

3. Select a settings panel.

From the list on the left side of the Target Settings window, select the settings panel that contains the settings you want to edit.

4. Modify the settings in the panel.

5. Click **OK**.

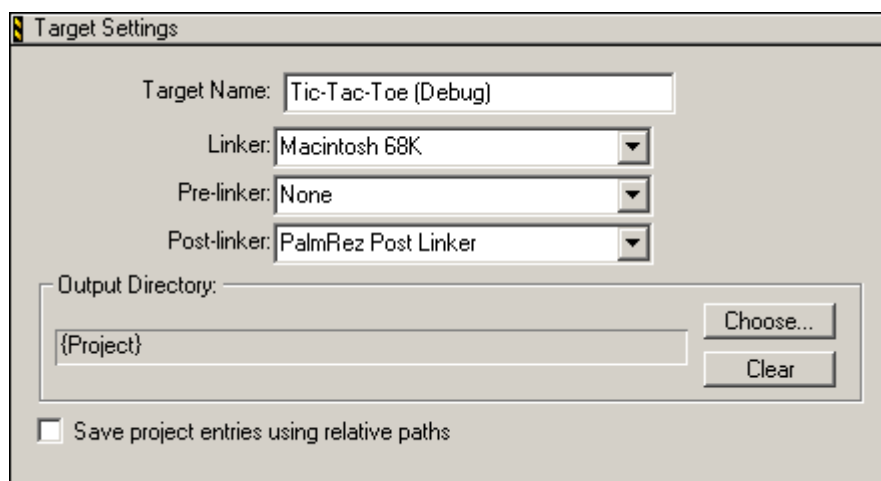
The IDE saves your changes and closes the Target Settings window.

Target Settings

The **Target Settings** panel (shown in [Figure 6.3](#)) is the most important settings panel in the Target Settings window.

When you choose a linker in this panel, you specify the target operating system and processor for the build target. The IDE shows and hides other settings panels in the Target Settings window based on your selection. Because the **Linker** setting affects the visibility of other settings panels, you should always set the linker first.

Figure 6.3 Target Settings panel



[Table 6.2](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

Table 6.2 Target Settings panel items

Item	Description
Target Name	This text field contains the name of the build target.
Linker	This menu determines which linker (and compiler) the IDE uses to build the target.
Pre-Linker	This menu determines which pre-linkers (if any) the IDE uses to build the target. Pre-linkers perform work on object code before the link stage of a build.
Post-Linker	This menu determines which post-linker (if any) the IDE uses to build the target. Post-linkers perform work on object code after the link stage of a build.

Table 6.2 Target Settings panel items

Item	Description
Output Directory	This directory is the folder where the IDE places the final output file when the IDE builds the target. Click Choose to specify another directory.
Save Project Entries Using Relative Paths	Check this box when you want to add two or more files with the same name to a project.

Target Name

Enter in the **Target Name** text field the name of the build target. The IDE displays this name in several places:

- the **Targets** page of the Project window
- the IDE **Edit > TargetName Settings** menu
- the Build Target Menu on the Project window

NOTE	The build target name is not the final output filename. It is simply the name by which you identify the build target when working in the CodeWarrior IDE.
-------------	---

Linker

This menu determines which linker (and compiler) the IDE uses to build the target. Select a linker from the items listed in this menu. [Table 6.3](#) describes each of the menu items in this menu.

Table 6.3 Linker menu item descriptions

Menu Item	Description
None	No linker is selected.
Macintosh 68K	<p>This linker uses the Macintosh 68K compiler and the PalmRez post-linker to create Palm OS applications.</p> <p>We provide this linker for compatibility with older Palm OS projects. If you create a new Palm OS project, you should use the Palm OS 68K linker instead of this one.</p>

Table 6.3 Linker menu item descriptions

Menu Item	Description
Palm OS ARMlet	This linker compiles C and C++ source code into ARM-native machine code. Use this linker to create ARMlets.
Mac OS Merge	This linker merges all Macintosh resources in the project into a single resource (<code>.rsrc</code>) file. This linker is handy for generating Constructor resource files from Rez (<code>.r</code>) source files.
Palm OS 68K	This linker is the preferred linker for creating Palm OS executables. All new Palm OS projects should use this linker. This linker does not require the PalmRez post-linker and does not generate Mac OS executables as a temporary step during a build.

Pre-Linker

This menu determines which pre-linker (if any) the IDE uses to build the target. Pre-linkers perform work on object code immediately before the link stage of a build

Select **Batch Runner Post-Linker** from this menu to have the IDE execute any batch (`.bat`) files in the target immediately before the link stage of the build. You might use this feature to copy output files to a special location or to pre-process a PRC file.

Post-Linker

This menu determines which post-linker (if any) the IDE uses to build the target. Post-linkers perform work on object code immediately after the link stage of a build

For Palm OS projects that use the Macintosh 68K linker, select **PalmRez Post-Linker** from this menu. The **PalmRez Post-Linker** converts object code into the Palm OS compatible Palm OS Resource Collection (`.prc`) format.

For Palm OS applications that use the Palm OS 68K linker, do not select **PalmRez Post-Linker** from this menu. You may select **None** or **Batch Runner Post-Linker** from this menu instead.

Select **Batch Runner Post-Linker** from this menu to have the IDE execute any batch (`.bat`) files in the target immediately after the link stage of the build. You might use this feature to copy output files to a special location or to post-process a PRC file.

Save Project Entries Using Relative Paths

Check this box when you want to add two or more files with the same name to a project.

When this box is checked, the IDE stores the location of project files as relative paths. When searching for project files, the IDE combines **Access Paths** settings with the stored relative path to find the files.

When this box is unchecked, the IDE stores only the file name of project files. Therefore, each project file must have a unique name. When searching for project files, the IDE combines **Access Paths** settings with the file name to find the files.

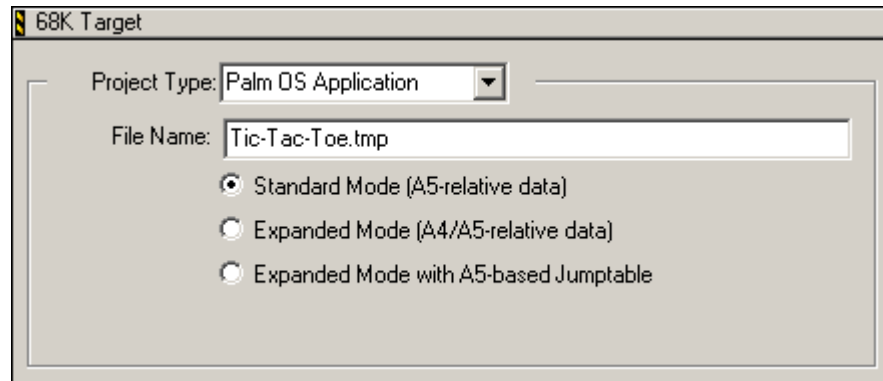
68K Target

This settings panel lets you set the type of project you want to create.

The IDE makes the **68K Target** settings panel (shown in [Figure 6.4](#)) available when you select **Macintosh 68K** from the **Linker** menu in the [Target Settings](#) panel.

NOTE	We provide this settings panel for compatibility with older Palm OS projects. If you want to create a new Palm OS project, you should use the Palm OS 68K Target settings panel instead of this one.
-------------	--

Figure 6.4 68K Target settings panel



The IDE displays a different pane in this settings panel based on the **Project Type** menu selection. Here are the possible **Project Type** menu selections:

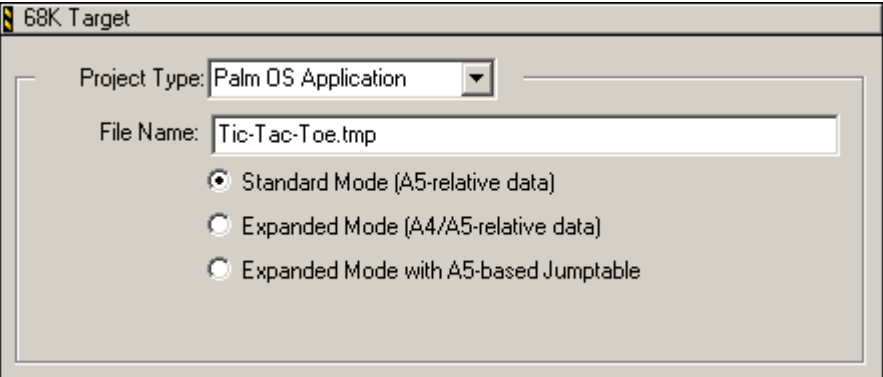
- [Palm OS Application](#)
- [Palm OS Code Resource](#)
- [Palm OS Static Library](#)
- [Mac OS Application](#)
- [Mac OS Code Resource](#)

Palm OS Application

The IDE displays the **Palm OS Application** pane (shown in [Figure 6.5](#)) when you select **Palm OS Application** from the **Project Type** menu in the [68K Target](#) panel. Select this project type to build a Palm OS application.

NOTE	We provide this settings panel for compatibility with older Palm OS projects. If you want to create a new Palm OS application, you should use the appropriate pane of the Palm OS 68K Target settings panel instead of this one.
-------------	--

Figure 6.5 68K Target: Palm OS Application pane



[Table 6.4](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

Table 6.4 68K Target: Palm OS Application pane items

Item	Description
Project Type	This menu determines the type of final output file the IDE creates when you build the target.
File Name	Enter in this text field the file name of the Palm OS application file the IDE generates when it builds the target.
Standard Mode (A5-relative data)	Select this option to have the application run in standard data storage mode with A5-relative data.

Table 6.4 68K Target: Palm OS Application pane items

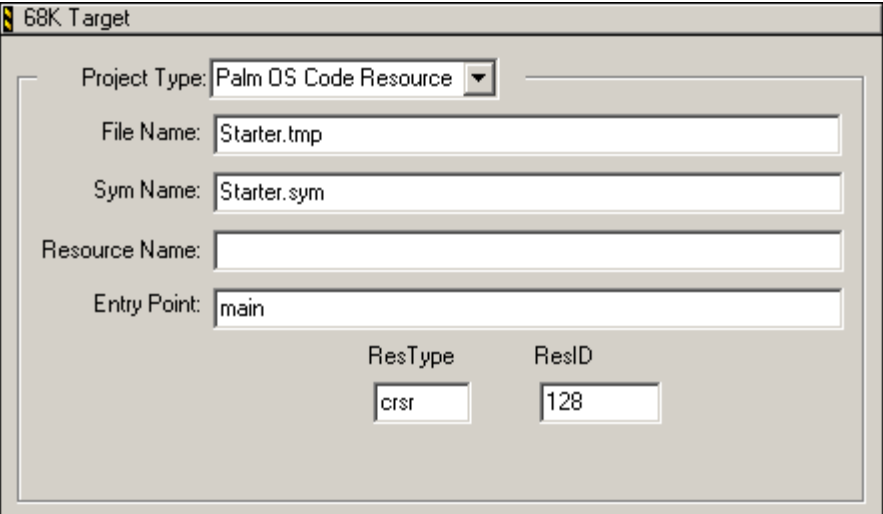
Item	Description
Expanded Mode (A4/A5-relative data)	Select this option to have the application run in expanded data storage mode with A4/A5-relative data. For more information on expanded mode, see “Expanded Global Space Mode” on page 198 .
Expanded Mode with A5-based Jumptable	Select this option to have the application run in expanded data storage mode with an A5-based jump table. For more information on expanded mode, see “Expanded Global Space Mode” on page 198 .

Palm OS Code Resource

The IDE displays the **Palm OS Code Resource** pane (shown in [Figure 6.6](#)) when you select **Palm OS Code Resource** from the **Project Type** menu in the [68K Target](#) panel. Select this project type to build a Palm OS code resource.

NOTE We provide this settings panel for compatibility with older Palm OS projects. If you want to create a new Palm OS code resource, you should use the appropriate pane of the [Palm OS 68K Target](#) settings panel instead of this one.

Figure 6.6 68K Target: Palm OS Code Resource pane



[Table 6.5](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

Table 6.5 68K Target: Palm OS Code Resource pane items

Item	Description
Project Type	This menu determines the type of final output file the IDE creates when you build the target.
File Name	Enter in this text field the name of the code resource file the IDE generates when it builds the target. The file name should end with the extension <code>.tmp</code> . The IDE passes this file to the PalmRez post-linker during a build.

Table 6.5 68K Target: Palm OS Code Resource pane items

Item	Description
Sym Name	Enter in this text field the name of the file in which the IDE places the code resource debugger symbolic information. The IDE creates this file in the project output folder when you build the target.
Resource Name	This option does not apply to Palm OS code resources.
Entry Point	Enter in this field the name of the function you want the Palm OS to call when it loads the code resource.
ResType	Enter in this field a four-character type code for the code resource.
ResID	Enter in this field a numerical resource ID for the code resource.

ResType

Enter in this field a four-character type code for the code resource. [Table 6.6](#) shows some typical values for this field.

Table 6.6 Typical resource type values

Resource Type	Description
libr	use for shared libraries
hack	use for Palm OS hacks

ResID

Enter in this field a numerical resource ID for the code resource. It may contain any value between 0 and 65535. Table shows some typical values for this field.

Table 6.7 Typical resource ID values

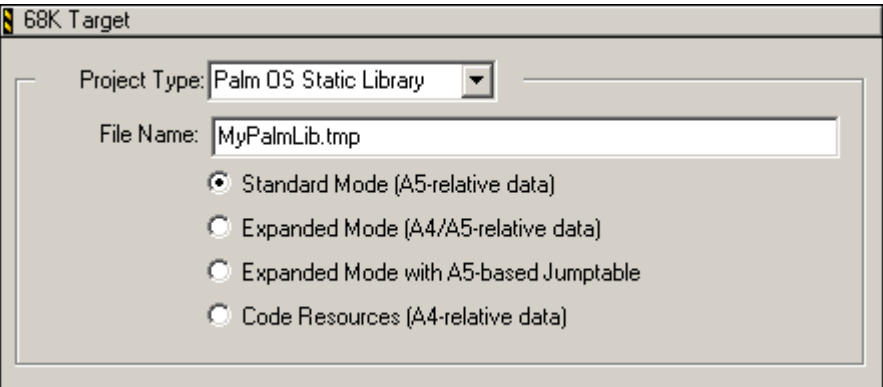
Resource Type	Description
0	use for shared libraries
<i>trap number</i>	use for Palm OS hacks

Palm OS Static Library

The IDE displays the **Palm OS Static Library** pane (shown in [Figure 6.7](#)) when you select **Palm OS Static Library** from the **Project Type** menu in the [68K Target](#) panel.

NOTE We provide this settings panel for compatibility with older Palm OS projects. If you want to create a new Palm OS static library, you should use the appropriate pane of the [Palm OS 68K Target](#) settings panel instead of this one.

Figure 6.7 68K Target: Palm OS Static Library pane



[Table 6.8](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

Table 6.8 68K Target: Palm OS Static Library pane items

Item	Description
Project Type	This menu determines the type of final output file the IDE creates when you build the target.
File Name	Enter in this text field the name of the static library file the IDE generates when it builds the target. The file name should end with the extension <code>.tmp</code> . The IDE passes this file to the PalmRez post-linker during a build.
Standard Mode (A5-relative data)	Select this option to have the application run in standard data storage mode with A5-relative data.
Expanded Mode (A4/A5-relative data)	Select this option to have the application run in expanded data storage mode with A4/A5-relative data. For more information on expanded mode, see “Expanded Global Space Mode” on page 198 .

Table 6.8 68K Target: Palm OS Static Library pane items

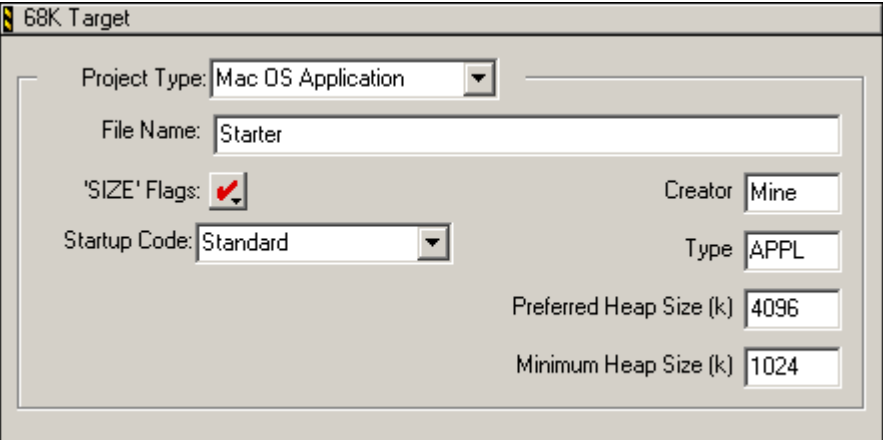
Item	Description
Expanded Mode with A5-based Jumptable	Select this option to have the application run in expanded data storage mode with an A5-based jump table. For more information on expanded mode, see “Expanded Global Space Mode” on page 198 .
Code Resources (A4-relative data)	Select this option to have the application run in code resource data storage mode with A4-relative data. If you intend to link the static library to a code resource, select this option.

Mac OS Application

The IDE displays the **Mac OS Application** pane (shown in [Figure 6.8](#)) when you select **Mac OS Application** from the **Project Type** menu in the [68K Target](#) panel.

NOTE This project type is provided for compatibility with projects created with the Mac OS-hosted version of CodeWarrior Development Tools for Palm OS that use the Mac OS-hosted Palm OS Simulator library.

Figure 6.8 68K Target: Mac OS Application pane



[Table 6.9](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

Table 6.9 68K Target: Mac OS Application pane items

Item	Description
Project Type	This menu determines the type of final output file the IDE creates when you build the target.
File Name	Enter in this text field the name of the Mac OS application file the IDE generates when it builds the target.
'SIZE' Flags	Select flags from this menu to apply to the Mac OS application file when the IDE builds the target.
Startup Code	Select from this menu the startup code the linker should add to the Mac OS application.
Creator	Enter in this field a four-character type code for the application.

Table 6.9 68K Target: Mac OS Application pane items

Item	Description
Type	Enter in this field a numerical resource ID for the application.
Preferred Heap Size (k)	Enter in this field the preferred (or maximum) amount of RAM, in kilobytes, that the operating system should allocate for your application.
Minimum Heap Size (k)	Enter in this field the minimum amount of RAM, in kilobytes, that the operating system must have available before it runs your application.

Startup Code

Select from this menu the startup code the linker should add to the Mac OS application. [Table 6.10](#) describes the items in this menu.

Table 6.10 Startup Code menu items

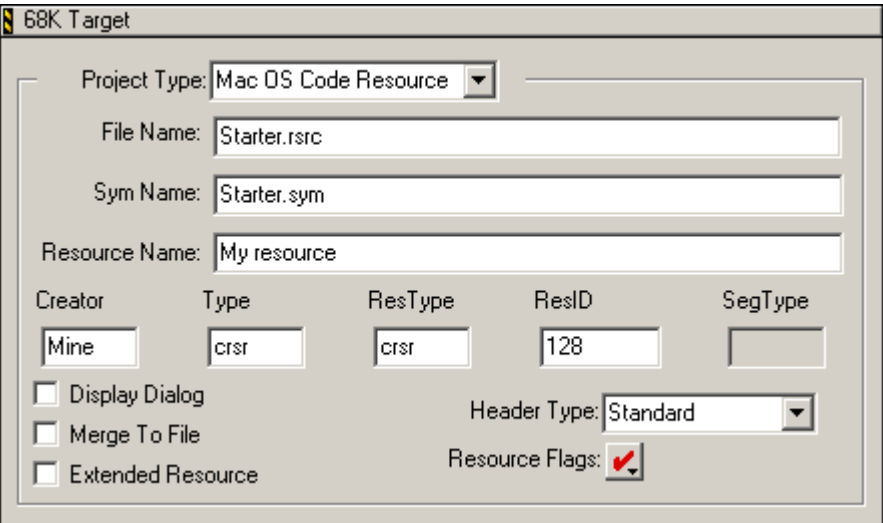
Menu Item	Description
Standard	Select this item to have the linker add the standard startup code for most Palm OS applications.
The Debugger-Aware	Select this item to use Steve Jasik's <i>TheDebugger</i> , a third-party debugger, to debug multi-segment applications.
Custom	Select this item to prevent the linker from adding startup code to the application. This allows you to create your own custom startup code.

Mac OS Code Resource

The IDE displays the **Mac OS Code Resource** pane (shown in [Figure 6.9](#)) when you select **Mac OS Code Resource** from the **Project Type** menu in the [68K Target](#) panel.

NOTE This panel is included for legacy projects only. If you want to create a Palm OS code resource, use the [Palm OS Code Resource](#) pane of the [Palm OS 68K Target](#) panel instead.

Figure 6.9 68K Target: Mac OS Code Resource pane



[Table 6.9](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

Table 6.11 68K Target: Mac OS Code Resource pane items

Item	Description
Project Type	This menu determines the type of final output file the IDE creates when you build the target.
File Name	Enter in this text field the name of the Mac OS code resource file the IDE generates when it builds the target.
Sym Name	Enter in this text field the name of the file in which the IDE places the code resource debugger symbolic information. The IDE creates this file in the project output folder when you build the target.
Resource Name	Enter in this field the resource name of the code resource.

Table 6.11 68K Target: Mac OS Code Resource pane items

Item	Description
Creator	Enter in this field a four-character type code for the code resource.
Type	Enter in this field a numerical resource ID for the code resource.
ResType	Enter in this field a four-character type code for the code resource.
ResID	Enter in this field a numerical resource ID for the code resource.
SegType	n/a
Display Dialog	Check this box to have the IDE ignore the File Name setting during a build and instead prompt you for the file name and location of the code resource file.
Merge To File	Check this box to have the IDE merge the code resource into the output file. Uncheck this box to have the IDE store the code resource in a new file.
Extended Resource	Check this box if the code resource is larger than 32 kilobytes in size. This option ensures that the operating system uses proper addressing mode when it accesses the code resource.
Header Type	Choose a header type from this menu to specify the type of code resource header the IDE places at the beginning of the code resource.
Resource Flags	Choose resource flags in this pop-up menu to set the resource attributes assigned to the code resource.

Resource Flags

Select resource flags from this pop-up menu to set the resource attributes for the code resource. [Table 6.12](#) lists each resource attribute.

Table 6.12 Resource Flag menu items

Resource Flag	Description
System Heap	Check this item to have the resource loaded into the system heap. uncheck this item to have the resource loaded into the calling application's heap.
Purgeable	Check this item to make the resource purgeable. Uncheck this item to make the resource nonpurgeable.
Locked	Check this item to make the resource nonpurgeable regardless of whether or not you checked the Purgeable item. Uncheck this item to make the resource purgeable or nonpurgeable depending on the value of the Purgeable item.
Protected	Check this item to make it so that your application cannot use Resource Manager routines to change the resource ID or resource name, modify the resource contents, or remove the resource from its resource fork. Uncheck this item to remove this protection.
Preload	Check this item to have the Resource Manager read the resource data into memory immediately after opening its resource fork. Uncheck this item to have the Resource Manager read the resource data into memory only when the application needs the data.

Palm OS 68K Target

This settings panel lets you set the type of project you want to create. The IDE makes the **Palm OS 68K Target** settings panel (shown in [Figure 6.10](#)) available when you select **Palm OS 68K** from the **Linker** menu in the [68K Target](#) panel.

NOTE This is the preferred settings panel for new Palm OS projects.

Figure 6.10 Palm OS 68K Target settings panel

Palm OS 68K Target

Target Type: 68K Application (Standard)

File Name: MyApp.prc

Code Entry Point: __Startup__

Code Resource Type: Multi-segment Code Resource

Code Resource ID: 0

Standard Resources

- ☒ Application Name ('tAIN'): MyApp
- ☒ Application Version ('tver'): 1.0
- ☒ Minimum Stack Size ('pref'): 1024
- ☐ Disallow Resource Overlays ('xprf')

The IDE displays a different pane in this settings panel based on the **Project Type** menu selection. Here are the possible **Project Type** menu selections:

- [68K Application \(Standard, Expanded, or Exp. A5-Jumptable\)](#)
- [68K Shared Library](#)
- [68K Code Resource](#)
- [Merged Resource File](#)
- [68K Static Library \(Standard, Expanded, or Exp. A5-Jumptable\)](#)

68K Application (Standard, Expanded, or Exp. A5-Jumptable)

The IDE displays one of the **68K Application** panes ([Figure 6.11](#)) when you select one of the **68K Application** items from the **Target Type** menu in the [Palm OS 68K Target](#) panel.

The 68K Application settings panels are identical, except for the following differences:

Select **68K Application (Standard)** to build a Palm OS 68K application that runs in standard data storage mode with A5-relative data.

Select **68K Application (Expanded)** to build a Palm OS 68K application that runs in expanded data storage mode with A4/A5-relative data.

Select **68K Application (Exp. A5-Jumptable)** to build a Palm OS 68K application that runs in expanded data storage mode with an A5-based jump table.

NOTE See [“Expanded Global Space Mode” on page 198](#) for more information about expanded mode.

Figure 6.11 Palm OS 68K Target: 68K Application (Standard) pane

Palm OS 68K Target

Target Type: 68K Application (Standard)

File Name: MyApp.prc

Code Entry Point: __Startup__

Code Resource Type: Multi-segment Code Resource

Code Resource ID: 0

Standard Resources

- ☒ Application Name ('tAIN'): MyApp
- ☒ Application Version ('tver'): 1.0
- ☒ Minimum Stack Size ('pref'): 1024
- ☐ Disallow Resource Overlays ('xprf')

[Table 6.13](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

Table 6.13 Palm OS 68K Target: 68K Application pane items

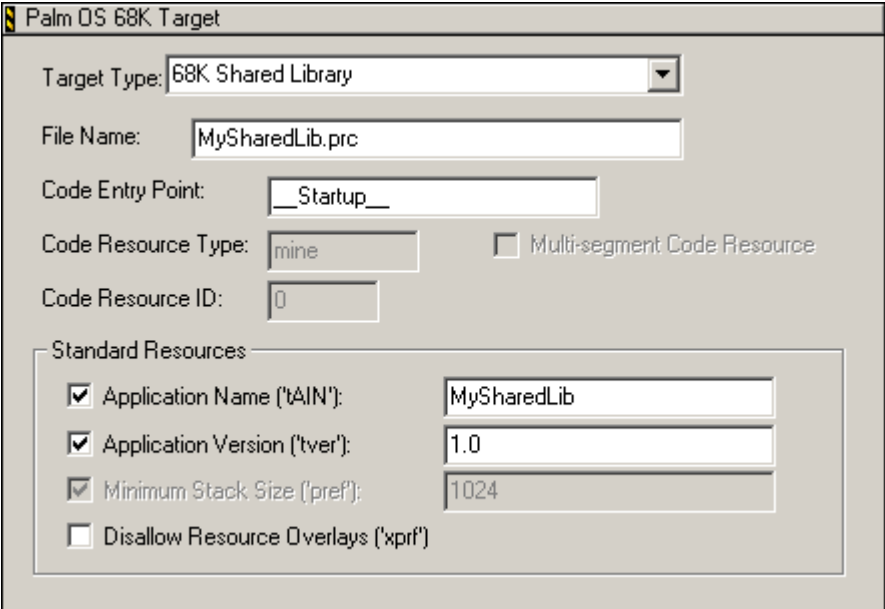
Item	Description
Target Type	This menu determines the type of final output file the IDE creates when you build the target.
File Name	Enter in this text field the name of the application file the IDE generates when it builds the target.
Code Entry Point	Enter in this field the name of the function you want Palm OS to call when it loads the application.
Code Resource Type	This option does not apply to Palm OS applications.
Code Resource ID	This option does not apply to Palm OS applications.
Multi-segment Code Resource	This option does not apply to Palm OS applications.
Application Name ('tAIN')	Check this box to have the linker add an application name ('tAIN') resource to the application file. Enter the name in the text field next to this option.
Application Version ('tver')	Check this box to have the linker add an application version ('tver') resource to the application file. Enter the version in the text field next to this option.
Minimum Stack Size ('pref')	Check this box to have the linker add an application size ('pref') resource to the application file. Enter the minimum stack size, in kilobytes, in the text field next to this option.
Disallow Resource Overlays ('xprf')	Check this box to have the linker add a 'xprf' resource to the application file that tells the operating system that it should not allow resource overlays.

68K Shared Library

The IDE displays the **68K Shared Library** pane (shown in [Figure 6.12](#)) when you select **68K Shared Library** from the **Target Type** menu in the [Palm OS 68K Target](#) panel.

Select this project type to build a Palm OS 68K shared library.

Figure 6.12 Palm OS 68K Target: 68K Shared Library pane



[Table 6.14](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

Table 6.14 Palm OS 68K Target: 68K Shared Library pane items

Item	Description
Target Type	This menu determines the type of final output file the IDE creates when you build the target.
File Name	Enter in this text field the name of the shared library file the IDE generates when it builds the target.
Code Entry Point	Enter in this field the name of the function you want Palm OS to call when it loads the shared library.
Code Resource Type	This option does not apply to Palm OS shared libraries.
Code Resource ID	This option does not apply to Palm OS shared libraries.

Table 6.14 Palm OS 68K Target: 68K Shared Library pane items

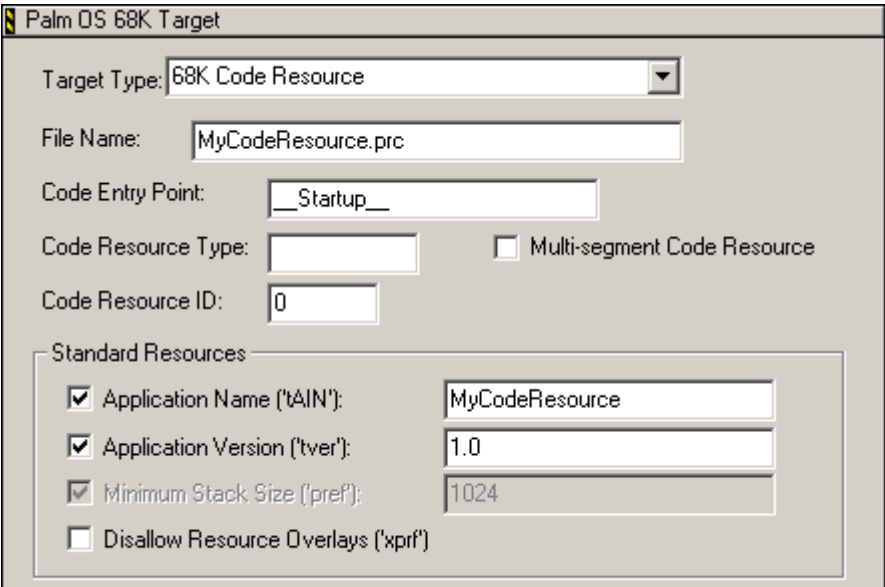
Item	Description
Multi-segment Code Resource	This option does not apply to Palm OS shared libraries.
Application Name ('tAIN')	Check this box to have the linker add an application name ('tAIN') resource to the shared library file. Enter the name in the text field next to this option.
Application Version ('tver')	Check this box to have the linker add an application version ('tver') resource to the shared library file. Enter the version in the text field next to this option.
Minimum Stack Size ('pref')	This option does not apply to Palm OS shared libraries.
Disallow Resource Overlays ('xprf')	Check this box to have the linker add a 'xprf' resource to the shared library file that tells the operating system that it should not allow resource overlays.

68K Code Resource

The IDE displays the **68K Code Resource** pane (shown in [Figure 6.13](#)) when you select **68K Code Resource** from the **Target Type** menu in the [Palm OS 68K Target](#) panel.

Select this project type to build a Palm OS 68K code resource. Developers typically use this project type to create hacks, system extensions, and application plug-ins.

Figure 6.13 Palm OS 68K Target: 68K Code Resource pane



[Table 6.15](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

Table 6.15 Palm OS 68K Target: 68K Code Resource pane items

Item	Description
Target Type	This menu determines the type of final output file the IDE creates when you build the target.
File Name	Enter in this text field the name of the code resource file the IDE generates when it builds the target.
Code Entry Point	Enter in this field the name of the function you want Palm OS to call when it loads the code resource.
Code Resource Type	Enter in this field the code section type for the code resource.

Table 6.15 Palm OS 68K Target: 68K Code Resource pane items

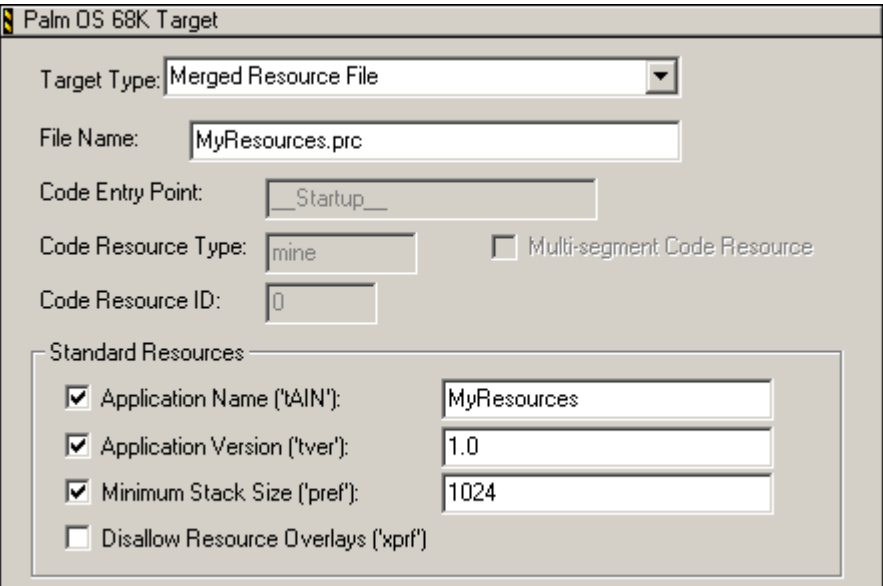
Item	Description
Code Resource ID	Enter in this field a number representing the first resource ID the IDE should use when it creates code resources in the build target. If the target output is multiple code resources, the IDE increments this number to generate the resource ID for subsequent code resources.
Multi-segment Code Resource	Check this box to have the IDE generate code resources that can span multiple segments.
Application Name ('tAIN')	Check this box to have the linker add an application name ('tAIN') resource to the code resource file. Enter the name in the text field next to this option.
Application Version ('tver')	Check this box to have the linker add an application version ('tver') resource to the code resource file. Enter the version in the text field next to this option.
Minimum Stack Size ('pref')	This option does not apply to Palm OS code resources.
Disallow Resource Overlays ('xprf')	Check this box to have the linker add a 'xprf' resource to the code resource file that tells the operating system that it should not allow resource overlays.

Merged Resource File

The IDE displays the **Merged Resource File** pane (shown in [Figure 6.14](#)) when you select **Merged Resource File** from the **Target Type** menu in the [Palm OS 68K Target](#) panel.

Select this project type to use the build target to merge PRC, resource object (.ro), and .bin files into a single PRC file without including code or data that the linker would normally generate. An example use for this would be to build a resource database for an overlay.

Figure 6.14 Palm OS 68K Target: Merged Resource File pane



[Table 6.16](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

Table 6.16 Palm OS 68K Target: 68K Code Resource pane items

Item	Description
Target Type	This menu determines the type of final output file the IDE creates when you build the target.
File Name	Enter in this text field the name of the merged resource file the IDE generates when it builds the target.
Code Entry Point	This option does not apply to merged resource files.
Code Resource Type	This option does not apply to merged resource files.

Table 6.16 Palm OS 68K Target: 68K Code Resource pane items

Item	Description
Code Resource ID	This option does not apply to merged resource files.
Multi-segment Code Resource	This option does not apply to merged resource files.
Application Name ('tAIN')	Check this box to have the linker add an application name ('tAIN') resource to the merged resource file. Enter the name in the text field next to this option.
Application Version ('tver')	Check this box to have the linker add an application version ('tver') resource to the merged resource file. Enter the version in the text field next to this option.
Minimum Stack Size ('pref')	Check this box to have the linker add an application size ('pref') resource to the merged resource file. Enter the minimum stack size, in kilobytes, in the text field next to this option.
Disallow Resource Overlays ('xprf')	Check this box to have the linker add a 'xprf' resource to the merged resource file that tells the operating system that it should not allow resource overlays.

68K Static Library (Standard, Expanded, or Exp. A5-Jumptable)

The IDE displays one of the **68K Static Library** panels (shown in [Figure 6.15](#)) when you select one of the **68K Static Library** items from the **Target Type** menu in the [Palm OS 68K Target](#) panel.

The 68K Static Library settings panels are identical, except for the following differences:

Select **68K Static Library (Standard)** to build a Palm OS 68K static library that runs in standard data storage mode with A5-relative data.

Select **68K Static Library (Expanded)** to build a Palm OS 68K static library that runs in expanded data storage mode with A4/A5-relative data.

Select **68K Static Library (Exp. A5-Jumptable)** to build a Palm OS 68K static library that runs in expanded data storage mode with an A5-based jump table.

NOTE See [“Expanded Global Space Mode” on page 198](#) for more information about expanded mode.

Figure 6.15 Palm OS 68K Target: 68K Static Library (Standard) pane

Palm OS 68K Target

Target Type: 68K Static Library (Standard)

File Name: MyLibrary.lib

Code Entry Point: __Startup__

Code Resource Type: Multi-segment Code Resource

Code Resource ID: 0

Standard Resources

- ☒ Application Name (<AIN>): Default App
- ☒ Application Version (<ver>): 1.0
- ☐ Minimum Stack Size (<pref>): 4096
- ☐ Disallow Resource Overlays (<xprf>)

[Table 6.17](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

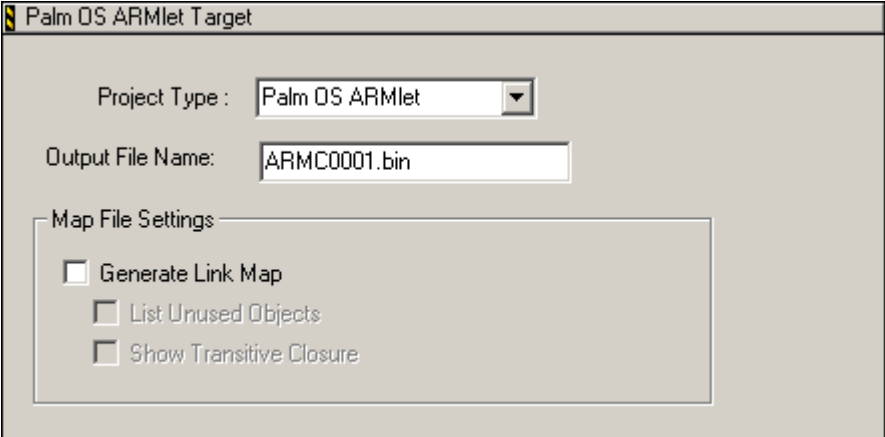
Table 6.17 Palm OS 68K Target: 68K Static Library pane items

Item	Description
Target Type	This menu determines the type of final output file the IDE creates when you build the target.
File Name	Enter in this text field the name of the static library file the IDE generates when it builds the target. Static library file names should end with the filename extension <code>.lib</code> .
Code Entry Point	This option does not apply to static libraries.
Code Resource Type	This option does not apply to static libraries.
Code Resource ID	This option does not apply to static libraries.
Multi-segment Code Resource	This option does not apply to static libraries.
Application Name ('tAIN')	This option does not apply to static libraries.
Application Version ('tver')	This option does not apply to static libraries.
Minimum Stack Size ('pref')	This option does not apply to static libraries.
Disallow Resource Overlays ('xprf')	This option does not apply to static libraries.

Palm OS ARMlet Target

The IDE makes the **Palm OS ARMlet Target** settings panel (shown in [Figure 6.16](#)) available when you select **Palm OS ARMlet** from the **Linker** menu in the [Target Settings](#) panel.

Figure 6.16 Palm OS ARMlet Target settings panel



The meaning of the items in this settings panel changes based on the **Project Type** menu selection. There are two items in the **Project Type** menu: **Palm OS ARMlet**, and **Palm OS Static Library**.

[Table 6.19](#) explains how the IDE uses the items for each project type. If more detailed information exists for an option, it is located below the table.

Table 6.18 Palm OS ARMlet Target panel items

Item	Description
Project Type	This menu determines the type of final output file the IDE creates when you build the target. Select Palm OS ARMlet to have the IDE build an ARMlet executable. Select Palm OS Static Library to have the IDE build an ARM static library.
Output File Name	Enter in this text field the name of the executable file you want the IDE to generate when you build the target. ARMlet executable binary files should have the filename extension <code>.bin</code> .
Generate Link Map	Check this box to have the linker generate a link map file, a text file that describes the contents of the compiled project.

Table 6.18 Palm OS ARMlet Target panel items

Item	Description
List Unused Objects	Check this box to have the linker list objects that is does not include in the link operation due to dead-stripping in the map file.
Show Transitive Closure	Check this box to have the linker add an ASCII tree diagram to the MAP file showing which link objects rely on other link objects.

Output File Name

Enter in this text field the name of the executable file you want the IDE to generate when you build the target.

ARMlet executable binary files should have the filename extension `.bin`.

If the ARMlet is linked against a build target that uses the [Palm OS 68K Linker](#), the name you enter here should be based on the resource type and ID you want the ARMlet to have. The first four characters should correspond to the resource type code. The next four characters should correspond to the hexadecimal representation of the resource ID. For example, if you want the ARMlet resource to have the resource type code 'ARMC', and to have the resource ID 1000, set the name to `ARMC03E8.bin`.

TIP You may add additional text to the filename after the first eight characters if you wish - the linker ignores everything after the first eight characters. For example, `ARMC0002_DrawSprite.bin` is a legal ARMlet file name.

Generate Link Map

Check this box to have the linker generate a link map file, a text file that describes the contents of the compiled project. The file includes lists of these items:

- symbols
- code segments
- routine and class sections
- global and local data sections
- source files that define data and routines

The linker places the text file in the project folder and names it the same name as the output file, but with the `.xMAP` filename extension.

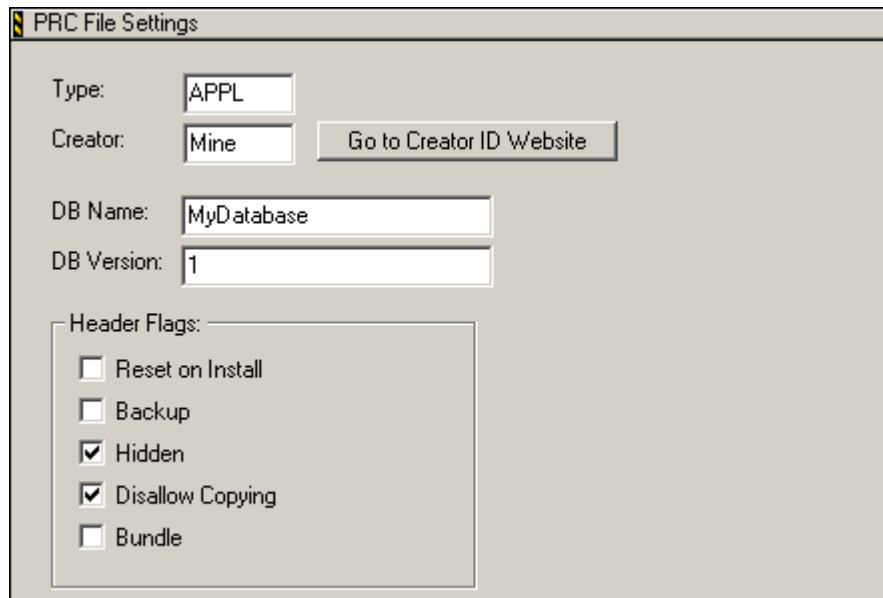
NOTE	Every routine listed in a code segment includes the name of the source code file in which it is defined. A missing source code filename implies that the routine is actually a reference to a linker-generated routine. Runtime libraries typically provide such linker-generated routines.
-------------	---

PRC File Settings

The IDE makes the **PRC File Settings** panel (shown in [Figure 6.17](#)) available when you select **Palm OS 68K** from the **Linker** menu in the [Target Settings](#) panel.

This settings panel lets you set various properties of the Palm OS Resource Collection (PRC) file the Palm OS 68K linker generates when you build the target.

Figure 6.17 PRC File Settings panel



[Table 6.19](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

Table 6.19 PRC File Settings panel items

Item	Description
Type	Enter in this field a four-character type code for the PRC file. Typically, applications are type 'appl', and shared libraries are type 'libr'.
Creator	Enter in this field a four-character creator code for the PRC file.
DB Name	Enter the name of the database. The linker adds this name in the database name field of the PRC file header.

Table 6.19 PRC File Settings panel items

Item	Description
DB Version	Enter an integer representing the version of the database. The linker adds this version to the version number field of the PRC file header. Palm OS uses this field to detect when you are trying to install an older version of a PRC file when a newer one exists.
Reset On Install	Check this box to have the linker set the reset bit in the PRC file header. When this bit is set, Palm OS resets the device after this file is installed. This is useful if the file is a system update or a driver that needs to hook into the operating system.
Backup	Check this box to have the linker set the backup bit in the PRC file header to indicate that you have modified the file. When this bit is set, HotSync software backs up this file to the user's computer the next time the user performs a HotSync operation.
Hidden	Check this box to have the linker set the hidden bit in the PRC file header. When this bit is set, the Palm OS application launcher does not display the file.
Disallow Copying	Check this box to have the linker set the copy prevention bit in the PRC file header. When this bit is set, you instruct Palm OS not to allow users to transfer the file between handheld devices with infrared communications or from memory to an expansion card.
Bundle	Check this box to have the linker set the bundle bit in the PRC file header. If this bit is set, the operating system copies any databases with the same creator ID as this file to memory when the user launches this file.

Creator

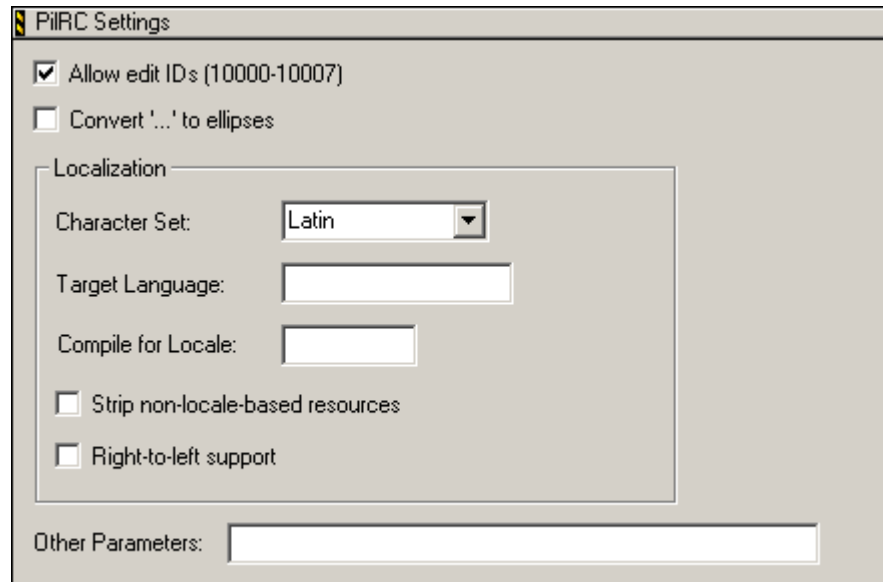
Enter in this field a four-character creator code for the resource collection file. Every application should have its own creator ID, registered with PalmSource. PalmSource reserves all creator IDs that are all lower-case letters for use by PalmSource and its licensees. Click **Go to Creator ID Website** to view the Palm OS Creator ID online database.

TIP	You should only use the 'STRT' creator ID for application development. Do not distribute applications with this creator ID.
------------	---

PiIRC Settings

The **PiIRC Settings** panel (shown in [Figure 6.18](#)) lets you control various aspects of the way the PiIRC resource compiler generates resource files when you build the target.

Figure 6.18 PiIRC Settings panel



[Table 6.20](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

Table 6.20 PiIRC Settings panel items

Item	Description
Allow edit IDs (10000-10007)	Check this box to have the PiIRC compiler allow the use of ID values that are reserved for the Edit menu. This option corresponds to the <code>-allowEditID</code> PiIRC command-line switch.
Convert '...' to ellipses	Check this box to have the PiIRC compiler convert the characters "..." to a single ellipses character ("...") whenever it encounters them during a build. This option corresponds to the <code>-noEllipsis</code> PiIRC command-line switch.

Table 6.20 PiIRC Settings panel items

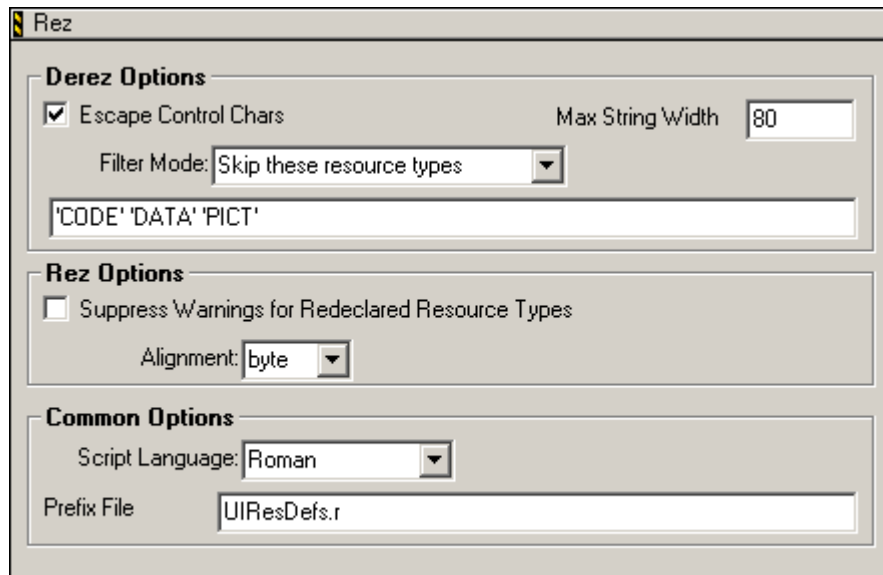
Item	Description
Character Set	Select a character set from this menu to have the linker use that character set font widths for AUTO width calculations. This option corresponds to these PiIRC command-line switches: <code>-Fh</code> , <code>-Fj</code> , <code>-F5</code> , <code>-Fg</code> , <code>-Fkm</code> , and <code>-Fkt</code> .
Target Language	Enter in this field the language you want the PiIRC compiler to use when it generates the resource files. This option corresponds to the <code>-L</code> PiIRC command-line switch.
Compile For Locale	Enter in this field the code representing the resource locales you want the PiIRC compiler to compile. This option corresponds to the <code>-Loc</code> PiIRC command-line switch.
Strip Non-Locale-Based Resources	Check this box to have the PiIRC compiler ignore resources that are not localizable. This option corresponds to the <code>-StripLoc</code> PiIRC command-line switch.
Right-To-Left Support	Check this box to have the PiIRC compiler enable right-to-left support for the Hebrew language.
Other Parameters	Enter in this field custom command-line options you want to pass to the PiIRC compiler when you build the target.

Rez

The IDE makes the **Rez** settings panel (shown in [Figure 6.19](#)) available when you select **Macintosh 68K** from the **Linker** menu in the [Target Settings](#) panel.

This settings panel lets you control how Rez compiles source files, and how Derez decompiles resource files when you build the target.

Figure 6.19 Rez settings panel



[Table 6.21](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

Table 6.21 Rez settings panel items

Item	Description
Escape Control Chars	Check this box to have Derez output escaped control characters in resource strings.
Max String Width	Enter in this text field the maximum desired string width of Derez output.
Filter Mode	Use this menu and text field to control which resource types Derez disassembles or ignores.
Suppress Warnings for Redeclared Resource Types	Check this box to have Rez suppress warnings when source code redeclares resource types.
Alignment	Use this menu to specify the alignment that Rez uses when writing a resource file.

Table 6.21 Rez settings panel items

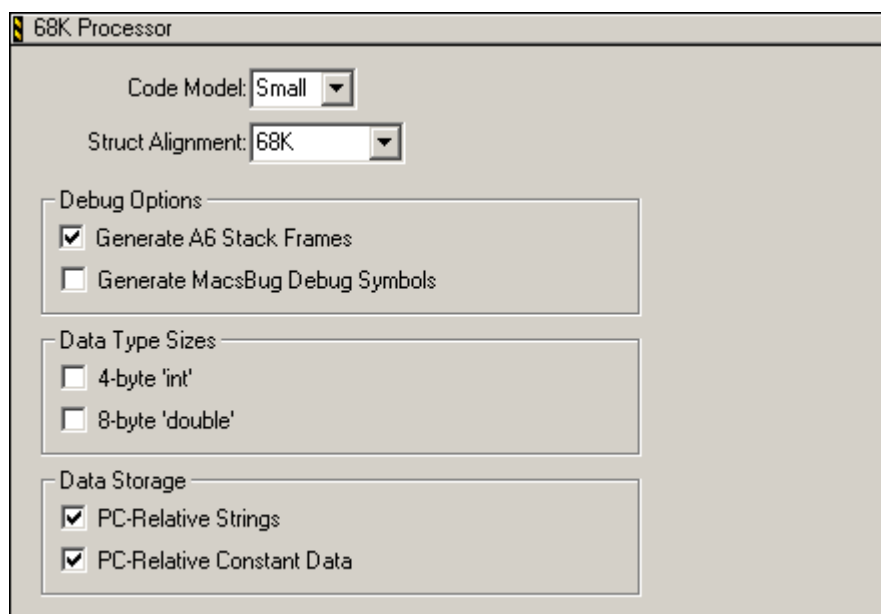
Item	Description
Script Language	Use this menu to control the script system that Rez uses to compile multi-byte characters.
Prefix File	Enter in this text field the file that the assembler reads before processing source files.

68K Processor

The IDE makes the **68K Processor** settings panel (shown in [Figure 6.20](#)) available when you select **Macintosh 68K** or **Palm OS 68K** from the **Linker** menu in the [Target Settings](#) panel.

This settings panel lets you control how the Macintosh 68K compiler generates code for Palm OS executables.

Figure 6.20 68K Processor settings panel



[Table 6.22](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

Table 6.22 68K Processor settings panel items

Item	Description
Code Model	Select an item from this menu to specify the access method for the data and code.
Struct Alignment	Select an item from this menu to specify the memory arrangement of records and structures. For all Palm OS development, select 68K .

Table 6.22 68K Processor settings panel items

Item	Description
Generate MacsBug Debug Symbols	Check this box to have the compiler generate debug symbols and store them in the executable code. This can greatly increase the size of executable code. Unlike the Palm Debugger and the Palm OS Emulator, the CW debugger does not use these debug symbols.
4-byte 'int'	Check this box to make values of the C/C++ <code>int</code> type 4 bytes in size (32 bits). Uncheck this box to make values of C/C++ <code>int</code> type 2 bytes in size (16 bits). This option corresponds to the <code>#pragma fourbyteints</code> directive. This setting effects the runtime libraries you should use to build the project. See "Runtime Libraries" on page 98 for more information.
8-byte 'double'	Check this box to make values of the C/C++ <code>double</code> data type 8 bytes in size (64 bits). For Palm OS development, this box should always be checked.
PC-Relative Strings	Check this box to have the linker store local-scoped string constants in the code segment. The compiler addresses these strings with PC-relative instructions. Checking this box can save global data space if your application uses a large number of string constants. This option also prevents you from exceeding the 64 KB space usually reserved for global data.
PC-Relative Constant Data	Check this box to have the compiler store constant data in the code segment. Checking this box lets you circumvent the 64KB data space limit, and leaves more space in the dynamic heap for <code>MemPtrNew</code> or <code>MemHandleNew</code> calls.

Code Model

Select an item from the **Code Model** menu to specify the access method for the data and code. [Table 6.23](#) describes the items in this menu.

Table 6.23 Code Model menu items

Menu Item	Description
Small	Select this model to have the linker use relative addressing (a 16-bit address) for all function calls. This selection is best for single-segment code resources and applications that are less than 32 KB in size, or for multi-segment code resources and applications in which all segments are less than 32 KB in size.
Smart	Select this model to have the linker use relative addressing (a 16-bit address) for function calls to functions within the same segment of code. For all other function calls, the linker uses absolute addressing (a 32-bit address).
Large	Select this model to have the linker use absolute addressing (a 32-bit address) for all function calls. This model is useful for targets that contain source files that generate more than 32K of code, or when the linker generates out-of-range link errors.

This option corresponds to these pragma directives:

- `#pragma far_code`
- `#pragma near_code`
- `#pragma smart_code`

For information on `#pragma` directives, refer to the *C Compilers Reference*.

TIP	If your Palm OS application or library runs on a device, use the small model, since it reduces the size of the code.
------------	--

PC-Relative Strings

The **PC-Relative Strings** check box lets you specify how the compiler and linker address string constants. This option corresponds to the `#pragma pcrelstrings` compiler directive.

Check this box to have the linker store local-scoped string constants in the code segment. The compiler addresses these strings with PC-relative instructions. Checking this box can save global data space if your application uses a large number of string constants. This option also prevents you from exceeding the 64 KB space usually reserved for global data.

Uncheck this box to have the linker store all the string constants in the global data segment.

NOTE	You should always use PC-relative strings in Palm OS shared libraries, hacks, and applications that handle special launch codes
-------------	---

Whether this box is checked or not, the compiler always stores global-scoped string constants in the global data segment. [Listing 6.1](#) shows an example of this. Also, the compiler always stores strings in C++ initialization code in the global data segment.

Listing 6.1 Global-scoped string constant storage

```
int x = f("Hello");    // "Hello" is in the global
                      // data segment

int foo()
{
    return f("World");  // "World" is in the local
                      //code segment
}
```

PC-Relative Constant Data

This check box lets you specify where the linker stores constant data. This option corresponds to the `#pragma pcrelconstdata` compiler directive.

Check this box to have the compiler store constant data in the code segment. Checking this box lets you circumvent the 64KB data space limit, and leaves more space in the dynamic heap for `MemPtrNew` or `MemHandleNew` calls.

Uncheck this box to have the compiler store constant data is stored in the global data segment.

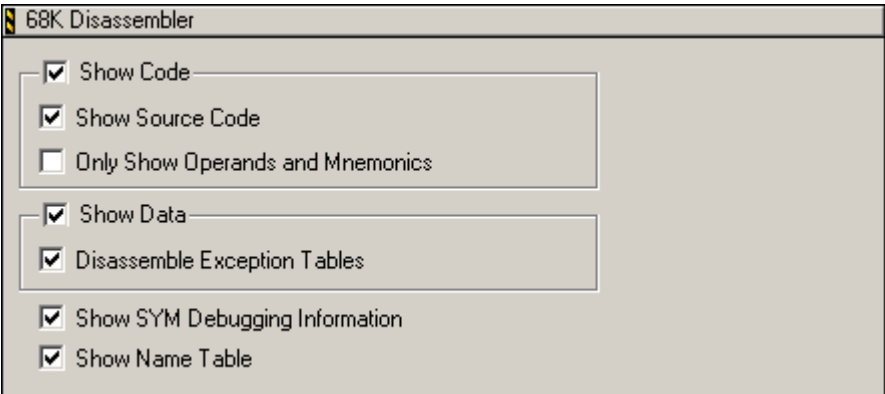
CAUTION	The restrictions of PC-Relative constant data may prevent some valid C and C++ code from compiling or linking: You cannot directly access PC-relative constant data from other code segments, nor can the PC-relative data contain any fields that hold the addresses of other data items.
----------------	--

68K Disassembler

The IDE makes the **68K Disassembler** settings panel (shown in [Figure 6.21](#)) available when you select **Macintosh 68K** or **Palm OS 68K** from the **Linker** menu in the [Target Settings](#) panel.

This settings panel lets you control the information that the IDE displays when you select **Project > Disassemble** from the CodeWarrior menu bar.

Figure 6.21 68K Disassembler settings panel



[Table 6.24](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

Table 6.24 68K Disassembler settings panel items

Item	Description
Show Code	Check this box to have the disassembler display object code in the disassembly listing.
Show Source Code	Check this box to have the disassembler display source code that corresponds to object code in the listing. The disassembler only displays source code for project files that contain debugger symbolic information.
Only Show Operands and Mnemonics	Check this box to have the linker hide assembly mnemonic hexadecimal values and operands. This makes it easier to edit disassembly data as assembly language source code.
Show Data	Check this box to have the disassembler display data sections in the disassembly listing.
Disassemble Exception Tables	Check this box to have the disassembler include data from exception tables in the disassembly listing.

Table 6.24 68K Disassembler settings panel items

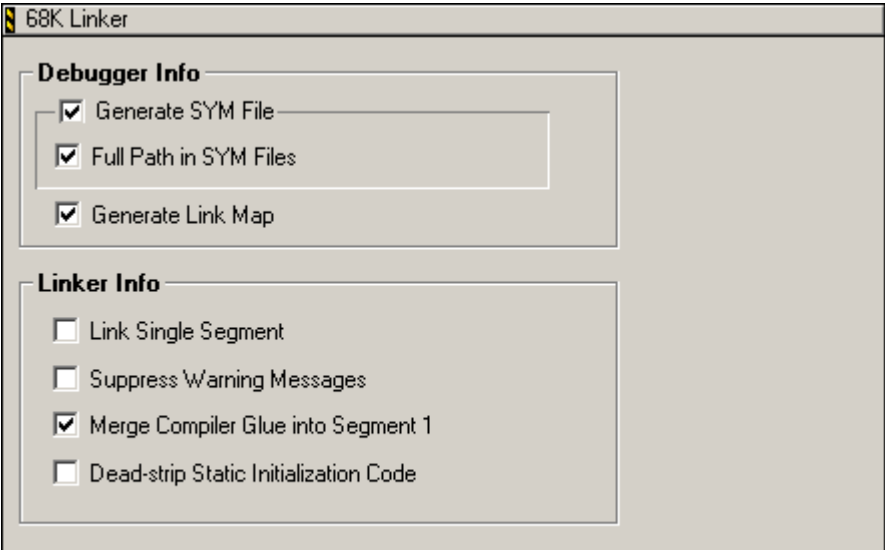
Item	Description
Show SYM Debugging Information	Check this box to have the disassembler display Palm OS debugger symbolic information in the disassembly listing.
Show Name Table	Check this box to have the disassembler display all object code symbols (in order of appearance) at the beginning of the disassembly listing.

68K Linker

The IDE makes the **68K Linker** settings panel (shown in [Figure 6.22](#)) available when you select **Macintosh 68K** or **Palm OS 68K** from the **Linker** menu in the [Target Settings](#) panel.

This settings panel lets you control how the linker links object code into final form.

Figure 6.22 68K Linker settings panel



[Table 6.25](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

Table 6.25 68K Linker settings panel items

Item	Description
Generate SYM File	Check this box to have the compiler and linker generate Palm OS debugger symbolic information for project files in the build target the next time you build it. Debugger symbolic information links source code in debugger windows to object code in memory. You can also use the <code>#pragma sym on</code> and <code>#pragma sym off</code> directives to control which functions have symbolic information.
Full Path in SYM Files	Check this box to have the linker place full path names in the debugger symbolic file. This option increases the debugger symbolic file size, but makes it easier for the debugger to locate source code files.

Table 6.25 68K Linker settings panel items

Item	Description
Generate Link Map	Check this box to have the linker generate a text file that describes the contents of the compiled project.
Link Single Segment	Check this box to have the linker ignore any #pragma segment directives in the source code and place all code into a single segment., allowing you to create single-segment applications. Uncheck this box to create multi-segment applications.
Suppress Warning Messages	Check this box to suppress all linker-specific warning messages.
Merge Compiler Glue Into Segment 1	Uncheck this box for Palm OS application targets. Check this box to have the linker place compiler initialization instructions and in-line C and C++ functions that cannot be called inline into the first code segment.
Dead-Strip Static Initialization Code	Check this box to have the linker remove all unreferenced runtime initialization and constructor calls from source files in the target. This option usually causes the linker to produce smaller and faster code.

Generate Link Map

Check this box to have the linker generate a link map file, a text file that describes the contents of the compiled project. The file includes lists of these items:

- symbols
- code segments
- routine and class sections
- global and local data sections
- source files that define data and routines

The linker places the text file in the project folder and names it the same name as the output file, but with the .MAP filename extension.

NOTE	Every routine listed in a code segment includes the name of the source code file in which it is defined. A missing source code filename implies that the routine is actually a reference to a linker-
-------------	---

generated routine. Runtime libraries typically provide such linker-generated routines.

Dead-Strip Static Initialization Code

Check this box to have the linker remove all unreferenced runtime initialization and constructor calls from source files in the target. This option usually causes the linker to produce smaller and faster code.

CAUTION Do not check this box if your program depends on executing static initializers.

For example:

A project contains the files `A.cpp`, `B.cpp`, and `C.cpp`. The `main()` function in `A.cpp` calls functions in `B.cpp` but not in `C.cpp`. `C.cpp` contains some global variables of types with constructors. The compiler generates static initializer code, which calls the constructors for the variables in `C.cpp`.

If this box is checked, the linker recognizes that no references are made to any of `C.cpp`'s variables. The linker therefore does not link the variables or the static initializers for them into the program.

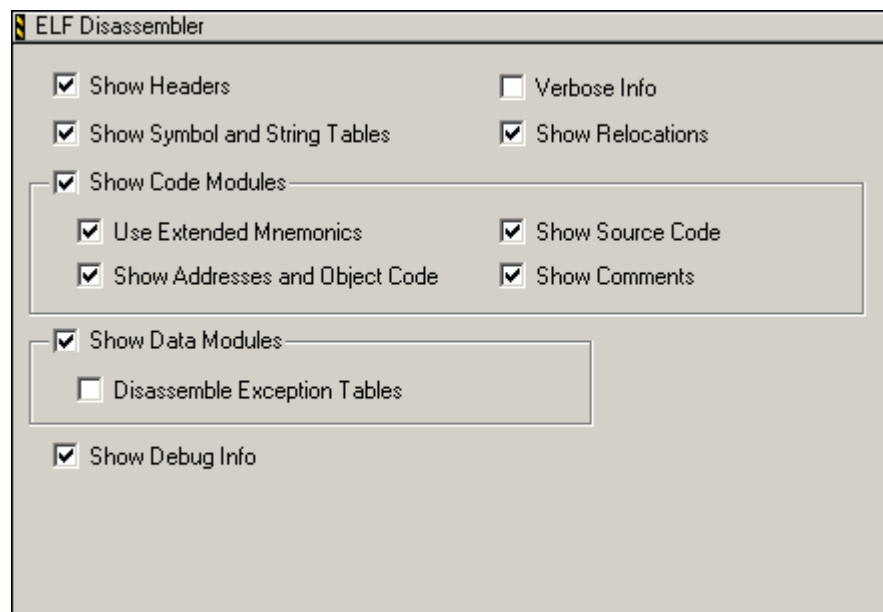
If the option is not enabled, the linker links the variables and the static initializers. The linker therefore may link variables and functions that the target never uses.

ELF Disassembler

The IDE makes the **ELF Disassembler** settings panel (shown in [Figure 6.23](#)) available when you select **Palm OS ARMIlet** from the **Linker** menu in the [Target Settings](#) panel.

This settings panel allows you to control settings related to the disassembly view the IDE displays when you disassemble Executable and Linker Format (ELF) object files.

Figure 6.23 ELF Disassembler settings panel



[Table 6.26](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

Table 6.26 ELF Disassembler settings panel items

Item	Description
Show Headers	Check this box to have the disassembler include ELF header information in the disassembled output.
Show Symbol and String Tables	Check this box to have the disassembler list the symbol table for the disassembled module.

Table 6.26 ELF Disassembler settings panel items

Item	Description
Verbose Info	Check this box to have the disassembler place addition information into the disassembled output, such as numeric equivalents for constants in the <code>.symtab</code> section, and unstructured hex dumps for <code>.line</code> , <code>.debug</code> , <code>extab</code> and <code>extabindex</code> sections.
Show Relocations	Check this box to have the linker output relocation information for the corresponding text (<code>.real.text</code>) or data (<code>.reala.data</code>) section.
Show Code Modules	Check this box to have the disassembler output the ELF code sections for the disassembled module.
Use Extended Mnemonics	Check this box to have the disassembler list the extended mnemonics for each instruction in the disassembled module.
Show Addresses and Object Code	Check this box to have the disassembler list the address and object code for the disassembled module.
Show Source Code	Check this box to have the disassembler list the source code for the disassembled module. The disassembler displays the source code in mixed mode with line number information from the original C source code.
Show Comments	Check this box to have the disassembler output comments produced by the disassembler in sections where comment columns exist.
Show Data Modules	Check this box to have the disassembler output ELF data sections (such as <code>.rodata</code> and <code>.bss</code>) for the disassembled module.
Disassemble Exception Tables	Check this box to have the disassembler output C++ exception tables for the disassembled module.
Show Debug Info	Check this box to have the disassembler output DWARF symbolic information in the disassembled output.

PalmRez Post-Linker

The IDE makes the **PalmRez Post-Linker** settings panel (shown in [Figure 6.24](#)) available when you select **PalmRez Post-Linker** from the **Post-Linker** menu in the [Target Settings](#) panel.

This settings panel lets you control how the PalmRez post-linker creates Palm OS PRC files from the temporary files that the Macintosh 68K compiler creates.

Figure 6.24 PalmRez Post-Linker settings panel

[Table 6.27](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

Table 6.27 PalmRez Post-Linker settings panel items

Item	Description
Mac Resource Files	Enter in this field the name of the file you want converted to a PRC file. You usually specify the same filename as in the File Name field of the 68K Target settings panel.
Text Description Files	Enter in this field the name of a resource descriptor file (a .r file) that describes Palm OS resources. Since this is the same format used by the Rez compiler, you can add the resource description file directly to the project instead of specifying it here if you want.
Output File	Enter the name you want the post-linker to give the PRC file.

Table 6.27 PalmRez Post-Linker settings panel items

Item	Description
Type	Enter in this field the four-character type code you want the post-linker to give the PRC file. The type code for Palm OS application should be 'appl'.
Creator	Enter in this field the four-character creator code you want the post-linker to give the PRC file. The creator code should differ from that of any other application. To ensure your creator code is unique, visit http://dev.palmos.com/creatorid/ .
Version	Enter in this field an integer representing the version number you want the post-linker to give the PRC file. If the user attempts to load two files with the same creator code onto a device, the HotSync software disposes of the copy with the lower version number.
Set Reset Bit	Check this box to have the post-linker set the reset bit in the PRC file header. When this bit is set, Palm OS resets the device after this file is installed. This is useful if the file is a system update or a driver that needs to hook into the operating system.
Set Backup Bit	Check this box to have the linker set the backup bit in the PRC file header to indicate that you have modified the file. When this bit is set, HotSync software backs up this file to the user's computer the next time the user performs a HotSync operation.
Set Hidden Bit	Check this box to have the linker set the hidden bit in the PRC file header. When this bit is set, the Palm OS application launcher does not display the file.
Set Copy-Prevention Bit	Check this box to have the linker set the copy prevention bit in the PRC file header. When this bit is set, you instruct Palm OS not to allow users to transfer the file between handheld devices with infrared communications or from memory to an expansion card.
Sort Resources By Size	Check this box to have the post-linker arrange resources in order of resource size.
Database Name	Enter the name of the database. The linker adds this name in the database name field of the PRC file header.
Transliteration	Select an item from this menu that represents the target operating system to specify how the post-linker translates text into the Palm OS character set. If a resource panel uses only 7-bit ASCII text, this setting has no effect.

Table 6.27 PalmRez Post-Linker settings panel items

Item	Description
Show Warnings	Check this box to have the post-linker report all post-linker-related warning messages.
Trap IDs	Enter in this field 4-digit hexadecimal values representing the trap vectors that your hack controls, separated by commas.

Palm OS Debugging

This settings panel lets you specify whether the debugger connects to a device, emulator, or simulator when you initiate a **Run** or **Debug** action, as well as various other parameters related to running and debugging applications.

The IDE makes the **Palm OS Debugging** settings panel (shown in [Figure 6.25](#)) available when you select **Macintosh 68K** or **Palm OS 68K** from the **Linker** menu in the [Target Settings](#) panel.

Figure 6.25 Palm OS Debugging settings panel

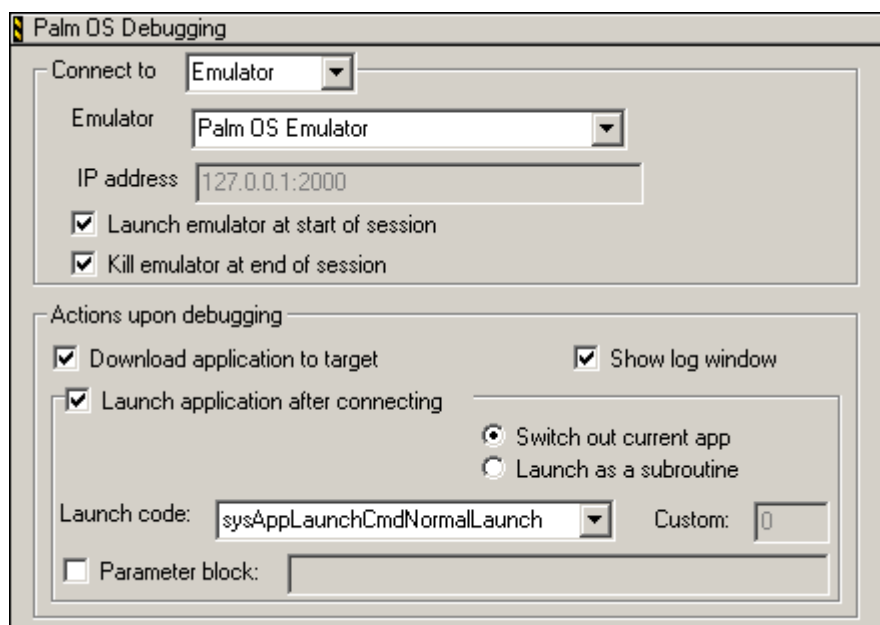
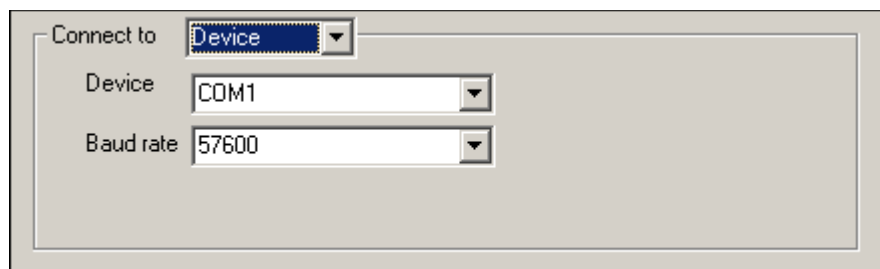


Figure 6.26 Palm OS Debugging - Connect to Device area



[Table 6.28](#) provides a short description of the items in this settings panel. If more detailed information exists for an option, it is located below the table.

Table 6.28 Palm OS Debugging settings panel items

Item	Description
Connect to	Select an item from this menu to specify whether you want the debugger to connect to an emulator or a device when you start a debug session in the IDE. When you select Device from this menu, the IDE displays the items show in Figure 6.26 .
Emulator	Select an item from this menu to specify the emulator the debugger should connect to when you start a debug session. The IDE displays this option when you select Connect to > Emulator . The items of this menu correspond to the information in this XML file: {CW}\bin\plugins\support\Palm_OS\PalmEmulators.xml.
IP Address	This read-only field displays the IP address associated with the emulator or simulator selected in the Emulator menu. The IDE displays this option when you select Connect to > Emulator .
Launch emulator at start of session	Check this box to have the debugger launch the emulator or simulator selected in the Emulator menu when you start the debug session. The IDE displays this option when you select Connect to > Emulator .
Kill emulator at end of session	Check this box to have the debugger kill the emulator or simulator selected in the Emulator menu when you end the debug session. The IDE displays this option when you select Connect to > Emulator .
Show log window	Check this box to have the debugger open a log window when you start a debug session in the IDE. The debugger outputs various types of information to the log window during the debug session.
Device	Select an item in this menu representing the communications port you want the debugger to use to connect to the device. The IDE displays this option when you select Connect to > Device .
Baud rate	Select an item in this menu representing the baud rate you want the debugger to use. The IDE displays this option when you select a device in the Device menu that supports baud rates.

Table 6.28 Palm OS Debugging settings panel items

Item	Description
Download application to target	Check this box to have the debugger download the application to the target emulator, simulator, or device when you start a debug session.
Launch application after connecting	Check this box to have the debugger launch the application after connecting to the emulator, simulator, or device.
Switch out current app	Select this option to have the debugger use the <code>SysUIAppSwitch()</code> system call to launch the application. This causes Palm OS to switch out the current application, then launch your application in its own context.
Launch as a subroutine	Select this option to have the debugger use the <code>SysAppLaunch()</code> system call to launch the application. This causes Palm OS to launch your application as a subroutine within the context of a special stub application the IDE downloads for this purpose.
Launch code	Select an item in this menu to specify the launch code that the debugger uses to start the application. The listed launch codes are the standard launch codes as defined in the <i>Palm OS Programmer's Companion</i> . To specify a launch code that is not in the list, select Custom , and enter the custom launch code number in the Custom field.
Custom	Enter in this field the custom launch code number you want the debugger to use to launch the application. This field is enabled when you select Custom from the Launch code menu.
Parameter block	Check this box to have the debugger pass a parameter block with the launch code when it launches the application. Enter the parameters in the text field next to the checkbox. The text field allows most standard C escape sequences such as <code>\n</code> and <code>\t</code> . To enter hexadecimal values, use the <code>\xHH</code> escape sequence (for example, <code>\xFF\xFF</code>).

C and C++ Compilers

This chapter explains how the compiler generates C and C++ code for the Palm OS® platform.

The sections in this chapter are:

- [Language Extensions](#)
- [Inline Assembly](#)
- [Number Formats](#)
- [Calling Conventions](#)
- [Variable Allocation](#)
- [Register Variables](#)
- [Floating Point Support](#)
- [Pragma Directives](#)
- [C++ Issues for Palm OS](#)
- [Expanded Global Space Mode](#)

NOTE

This chapter contains references to *K&R A*. These references refer to Appendix A, “Reference Manual,” of *The C Programming Language, Second Edition* (Prentice Hall) by Kernighan and Ritchie, where you can find more information on the current topic.

For example, the following reference shows that you can get more information in *The C Programming Language, Second Edition*, Appendix A, section 8.6.3 and section 10.1: For more information, see K&R, A8.6.3, A10.1.

[Table 7.1](#) shows you where to find information about code generation in other CodeWarrior manuals.

Table 7.1 More compiler and linker documentation

Topic	Document
CodeWarrior implementation of the C and C++ languages	<i>C Compilers Reference</i> manual
Information about CodeWarrior command-line compilers and linkers for Palm OS software development	<i>C Compilers Reference</i> manual
C/C++ Language and C/C++ Warnings target settings panels	<i>C Compilers Reference</i> manual
Controlling the size of C++ machine code	<i>C Compilers Reference</i> manual
Using compiler pragma directives	<i>C Compilers Reference</i> manual
Initiating builds, controlling file compilations, handling error reports	<i>IDE User's Guide</i>
Information about a particular error	<i>Error Reference</i> manual

Language Extensions

This section explains the Palm OS-specific extensions to the C and C++ standards found in the CodeWarrior C/C++ compiler.

You can disable some of these extensions with options in the **C/C++ Language** settings panel. See the *C Compilers Reference* for a complete description of the **C/C++ Language** settings panel.

Inline Data

When targeting 68K, the C/C++ compiler lets you include simple inline data with the `asm` declaration. Use the following syntax:

```
asm { constant, constant, . . . }
```

A *constant* can be a numeric constant or a string literal. For an example of inline data and resulting assembly code, see [Listing 7.1](#) and [Listing 7.2](#).

Listing 7.1 Inline data example

```
void foof()
{
    asm ( (short)0x4e71, (short)0x4e71 );

    // two 68K NOP instructions
    asm { 0x4e714e71, 0x4e714e71 };

    // four 68K NOP instructions
    asm ((char)'C', (char)'o', (short)'de', "Warrior");
}
```

[Listing 7.1](#) produces the following assembly code.

Listing 7.2 Assembly code from inline data

```
LINK      A6, #$0000
NOP
NOP
NOP          ; First two NOPs
NOP
NOP
NOP
NOP          ; Next four NOPs
```

```
DC.B      "CodeWarrior\0"  
UNLK      A6  
RTS
```

Specifying the Registers for Arguments

The C/C++ compiler lets you specify what registers a function should use for its parameters and return value. A function may use registers D0 through D2 and A0 through A1.

If you want to specify the registers to be used for arguments, you must specify the registers by using the `#pragma parameter` statement before declaring the function. Specify the registers in the argument list when you define the function.

This is the syntax for the `#pragma parameter`:

```
#pragma parameter return-reg func-name(param-regs)
```

The compiler passes the parameters for the function *func-name* in the registers specified in *param-regs* instead of the stack, and returns any return value in the register *return-reg*. Both *return-reg* and *param-regs* are optional.

For example, [Listing 7.3](#) shows the declaration and definition of a function that passes *a* in D0, *p* in A1, *x* in FP0, and *f* on the stack. The return value is in D2.

Listing 7.3 Using registers with functions

```
// prototype  
#pragma parameter __D2 function(__D0,__A1,__FP0)  
short function(long a, Ptr p, long double x, short f);  
  
// function definition  
short function(long a:__D0, Ptr p:__A1,  
               long double x:__FP0, short f) :__D2  
{  
    // ...  
}
```

For more information on function declarations and function definitions, see K&R, A8.6.3, A10.1.

Inline Assembly

This section explains how the CodeWarrior compiler supports inline assembly language programming.

NOTE This section does not document all the instructions available in Palm OS® platform assembly language. For information on Palm OS assembly language instructions, see *M68000 Family Programmer's Reference Manual* in this location (where *CWFolder* is the folder where you installed the CodeWarrior Development Tools for Palm OS package):

`CWFolder\CW for Palm OS Support\Documentation\Moto
rola CPU Docs\M68000PRM.pdf`

This section describes these topics:

- [Inline Assembly Instructions](#)
- [Inline Assembly Syntax](#)
- [Inline Assembly Directives](#)

Inline Assembly Instructions

The inline assembler uses the standard MC68000 assembler instructions as described in the *M68000 Family Programmer's Reference Manual*. The inline assembler accepts some additional directives described in [“Inline Assembly Directives” on page 187](#).

NOTE If you know the opcode for an unsupported assembly instruction, you can include it in your function with the `opword` directive, described in [“opword” on page 189](#).

Inline Assembly Syntax

This section explains how to use the CodeWarrior compiler's inline assembly support for assembly language programming, including assembler syntax.

Statements

All statements must follow this syntax:

```
[LocalLabel:] (instruction | directive) [operands]
```

Each instruction must end with a new line or a semicolon (;).

Hexadecimal constants are in C-style. For example:

```
move.l    0xABCDEF, d5
```

Assembler directives, instructions, and registers are not case-sensitive. For example these two statements are same:

```
move.l    b, D0
```

```
MOVE.L    b, d0
```

To specify that a block of code in your file should be interpreted as assembly language, use the `asm` keyword.

To ensure that the C/C++ compiler recognizes the `asm` keyword, you must turn off the **ANSI Keywords Only** option in the C/C++ Language panel. This panel and its options are fully described in the *C Compilers Reference*.

The assembly instructions are the standard Palm OS instruction mnemonics.

You use the Palm OS inline assembler to specify that an *entire function* is in assembly language. The CodeWarrior compiler also supports whole blocks of assembly statements embedded within a function.

[Listing 7.4](#) shows valid examples of inline assembly code for Palm OS:

Listing 7.4 Function-level inline assembly example

```
long int b;
struct mystruct {
    long int a;
} ;

static asm long f(void)          // Legal asm qualifier
{
    move.l    struct(mystruct.a)(A0),D0 // Accessing a struct.
```

```
add.l    b,D0    // Using a global variable, put return value
                // in D0.
rts      // Return from the function:
                // result = mystruct.a + b
}
```

Statement Level Inline Assembly Example

```
long square(short a)
```

```
{
    asm {
        move.w  a,d0    // fetch function argument 'a'
        mulu.w  d0,d0    // multiply
        return      // return from function (result is in D0)
    }
}
```

NOTE	The assembler never optimizes assembly language functions, regardless of compiler settings.
-------------	---

Labels

A label must end in a colon and may contain the @ character. See [Listing 7.5](#) for an example:

Listing 7.5 Example labels

```
asm void f(void)
{
    x1:  dc.b  "Hello world!\n"
    @x2: dc.w  5
}
```

Comments

You cannot begin comments with a semicolon (;), but you can use C and C++ comments. See [Listing 7.5](#) for an example:

Listing 7.6 Comment example

```
add.l    d5,d5
```

Preprocessor Features

You can use all preprocessor features, such as comments and macros, in the assembler. Just keep these points in mind when writing a macro definition:

- End each assembly statement with a semicolon (;), because the preprocessor ignores new lines. For example:

```
#define MODULO(x,y,result)\
move.w    x,D0; \
ext.l     D0; \
divs.w    y,D0; \
swap     D0; \
move.w    D0,result
```

- Use (%) instead of (#), as the preprocessor uses (#) as an operator to concatenate tokens. For example:

```
#define ClearD0    moveq %0,D0
```

Structures

You can refer to a field in a structure with the `struct` construct, as shown below:

```
struct(structTypeName.fieldName) structAddress
```

This instruction moves into D0 the `refCon` field in the `WindowRecord` that A0 points to:

```
move.l    struct(WindowRecord.refCon) (A0), D0
```

Global Variables

To refer to a global variable, just use its name, as shown in Listing:

Listing 7.7 Global variable example

```
int x;
asm void f(void)
{
    move.w    x,d0    // Moving x into d0
    // . . .
}
```

Local Variables and Arguments

The compiler changes the way it processes local variables and arguments depending on whether you use function-level or statement-level inline assembly code.

- [Function-Level](#)
- [Statement-Level](#)

Function-Level

The function-level inline assembler gives you two ways to refer to local variables and function arguments: you can do the work on your own or let the inline assembler do the work for you. To do it on your own, you must explicitly save and restore processor registers and local variables when entering and leaving your inline assembly function. You cannot refer to the variables by name. You can refer to function arguments off the stack pointer.

For example, this function moves its argument into d0:

```
asm void foo(short n)
{
    move.w      4(sp),d0 //  n
    // . . .
}
```

To let the inline assembler do it for you, use the directives `fralloc` and `frfree`. Declare your variables as you would in a normal C function. Then use the `fralloc` directive. It makes space on the stack for the local stack variables and reserves registers for the local register variables (with the statement `link #x,a6`). In your assembly code, you can refer to the local variables and variable arguments by name. Finally, use the `frfree` directive to free the stack storage and restore the reserved registers.

[Listing 7.8](#) is an example of using local variables and function arguments in function-level inline assembly.

Listing 7.8 Function-level Local Variables and Function Arguments

```
static asm short f(short n)
{
    register short a; // Declaring a as a register variable
    short b;         // and b as a stack variable
    // Note that you need semicolons after these statements.

    fralloc +        // Allocate space on stack and reserve registers.
```

```
move.w  n,a    // Using an argument and local var.
add.w   a,a
move.w  a,D0

frfree           // Free the space that fralloc allocated
rts
}
```

Statement-Level

Statement-level inline assembly allows full access to local variables and function arguments without using `fralloc` or `frfree` directives.

[Listing 7.9](#) is an example of using local variables and function arguments in statement-level inline assembly.

Listing 7.9 Statement-level local variables and function arguments.

```
long square(short a)
{
    long result=0;
    asm {
        move.w  a,d0      // fetch function argument 'a'
        mulu.w  d0,d0      // multiply
        move.l  d0,result // store in local 'result' variable
    }
    return result;
}
```

Returning From a Routine

You should end every inline assembly function with a return statement. This is an important rule to follow, because if you forget your return statement, the compiler neither adds one for you nor issues a warning message.

Use the `rts` return statement for ordinary C functions. For example:

```
asm void g(void)
{   add.l      d4, d5
    rts}
```

For statement-level returns, see [“return” on page 189](#) and [“naked” on page 189](#).

Inline Assembly Directives

This section describes some special assembler directives that the Palm OS inline assembler accepts. They are:

dc

`dc[(b|w|l)] constexpr (,constexpr)*`

This inline assembler directive defines a block of constant expressions, as initialized bytes, words, or long words. If there is no qualifier, the compiler assumes `dc.w`. For `dc.b` you can specify any string constant. For `dc.w` you can specify any 16-bit relative offset to a local label. For example:

```
asm void f(void)
{
    x1: dc.b  "Hello world!\n" // Creating a string
    x2: dc.w  1,2,3,4          // Creating an array
    x3: dc.l  3000000000        // Creating a number
}
```

ds

`ds[(b|w|l)] size`

This inline assembler directive defines a block of *size* bytes, words, or longs. The compiler initializes this block with null characters. If there is no qualifier, the compiler assumes `ds.w`. For example, this statement defines a block big enough for the structure `DRVHeader`.

```
ds.b  sizeof(DRVHeader)
```

entry

`entry [extern|static] name`

This inline assembler directive defines an entry point into the current function. Use the `extern` qualifier to declare a global entry point. Use the `static` qualifier to declare a local entry point. If you leave out the qualifier, the compiler assumes an `extern` qualifier.

```
static long MyEntry(void);
static asm long MyFunc(void)
```

```
{
    move.l    a,d0
    bra.s     L1

    entry     static MyEntry
    move.l    b,d0
L1: rts
}
```

fralloc

`fralloc [+]`

This inline assembler directive lets you declare local variables in an assembly function. The `fralloc` directive makes space on the stack for your local stack variables. The compiler reserves registers for your local register variables with the statement `link #x,a6`. For more information, see [“Local Variables and Arguments” on page 185](#).

The `fralloc` directive (without a +), pushes modified registers onto the stack.

The `fralloc +` directive pushes all register arguments into their Palm OS registers.

frfree

`frfree`

This inline assembler directive frees the stack storage area. The compiler restores the `fralloc`-reserved registers with the statement `unlk a6`. For more information, see [“Local Variables and Arguments” on page 185](#).

machine

`machine number`

This inline assembler directive defines the compiler’s code-coverage. For Palm OS development, the *number* must be 68000. No other value is valid for Palm OS development.

opword

`opword const-expr (, const-expr) *`

This inline assembler directive lets you specify instructions by their opcodes. This directive is similar to `dc.w`, but emphasizes that the expression is an instruction. For example, this directive calls `WaitNextEvent()`:

```
opword 0xA860      // WaitNextEvent
```

return

`return value`

This inline assembler directive lets you insert a compiler-generated stackframe cleanup and return instruction sequence with support for 68K calling conventions.

naked

`naked`

This inline assembly directive suppresses the compiler-generated stackframe setup and cleanup code. Functions with the `naked` directive cannot access local variables by name, and should not contain C code that implicitly or explicitly uses local variables or memory.

Make sure that your code does not fall off the end of your function by either supplying your own return instruction, or implementing some other form of flow control.

Listing 7.10 Using the naked directive

```
long square(short)
{
    asm{
        naked                // no compiler-generated stackframe or
                             // clean-up
        move.w 4(sp),d0      // fetch function argument from stack
        mulu.w d0,d0         // multiply
        rts                  // return from function (result in D0)
    }
}
```

Number Formats

This section explains how the CodeWarrior C/C++ compilers implement integer and floating-point types for Palm OS processors. You can also read `limits.h` for more information on integer types, and `float.h` for more information on floating-point types.

The topics in this section are:

- [Integer Formats](#)
- [68K Floating Point Formats](#)

Integer Formats

The 68K back-end compiler lets you choose the number of bytes allocated for an `int` using the **4-Byte Ints** option in the [68K Processor](#) panel. Using 4-byte integers decreases the execution speed of your code by increasing the potential for internal promotions and demotions.

[Table 7.2](#) shows the size and range of the integer types available when targeting the Palm OS.

Table 7.2 Palm OS integer types

Type	Option	Size	Value Range
bool	n/a	8 bits	true or false
char	Use Unsigned Chars is <i>not enabled</i> in the C/C++ Language panel	8 bits	-128 to 127
char	Use Unsigned Chars is <i>enabled</i> in the C/C++ Language panel	8 bits	0 to 255
signed char	n/a	8 bits	-128 to 127
unsigned char	n/a	8 bits	0 to 255
short	n/a	16 bits	-32,768 to 32,767
unsigned short	n/a	16 bits	0 to 65,535

Table 7.2 Palm OS integer types

Type	Option	Size	Value Range
int	4-Byte Ints is <i>not enabled</i> in the 68K Processor panel	16 bits	-32,768 to 32,767
	4-Byte Ints is <i>enabled</i> in the 68K Processor panel	32 bits	-2^{31} to $2^{31}-1$
unsigned int	4-Byte Ints is <i>not enabled</i> in the 68K Processor panel	16 bits	0 to 65,535
	4-Byte Ints is <i>enabled</i> in the 68K Processor panel	32 bits	0 to $2^{32}-1$
long	n/a	32 bits	-2^{31} to $2^{31}-1$
unsigned long	n/a	32 bits	0 to $2^{32}-1$
long long	n/a	64 bits	-2^{63} to $2^{63}-1$
unsigned long long	n/a	64 bits	0 to $2^{64}-1$

68K Floating Point Formats

The 68K back-end compiler lets you choose the number of bytes allocated for a `double` using the **8-Byte Doubles** option in the **68K Processor** settings panel. *For Palm OS development, enable the **8-byte Doubles** option.*

Table 7.3 Palm OS 68K floating point types

Type	Option	Size	Value Range
float	n/a	32 bits	1.17549e-38 to 3.40282e+38
short double	n/a	64 bits	2.22507e-308 to 1.79769e+308
double	8-Byte Doubles is on	64 bits	2.22507e-308 to 1.79769e+308
long double	n/a	80 bits	3.362103e-4932 to 1.18973e+4932

Calling Conventions

This section describes the C/C++ calling conventions for Palm OS development.

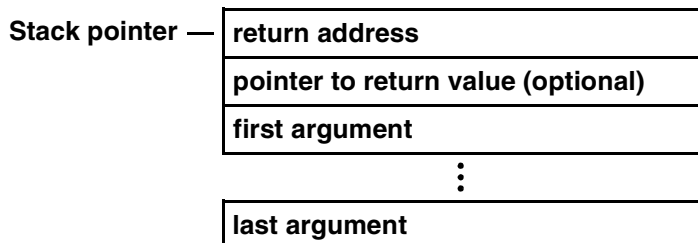
The compiler passes all the parameter values that are on the stack in reverse order.

The compiler passes the return value in different locations, depending on the nature of the value and compiler settings.

- It returns an integer value in register D0.
- It returns a pointer value in register A0.
- If it returns a value of any other type, the caller reserves a temporary storage area for that type in the caller's stack and passes a pointer to that area as the last argument. The function that has been called returns its value in the temporary storage area.

[Figure 7.1](#) depicts the contents of the stack in which you have called a C function.

Figure 7.1 Contents of the stack after calling a C function



Variable Allocation

This section describes how the C/C++ compiler allocates space for variables. The C/C++ compiler lets you declare structs and arrays to be any size. However, it does place some limits on how you allocate space for them. The Palm OS may further limit the size of variable allocations.

A function cannot contain more than 32KB of local variables. To avoid this problem, do one of the following:

- Dynamically allocate large variables.
- Declare large variables to be `static`. Note that you may run into the 32K limit on global variables.

Bitfields can be no larger than 32 bits.

See also, K&R, A4.3, A8.3, A8.6.2.

Register Variables

The compiler uses these registers for local variables:

- A2 through A4 for pointers (only A2 and A3 in expanded mode)
- A6 for pointers (if you disable A6 stack frames)
- D3 through D7 for integers and pointers

The compiler automatically allocates local variables and parameters according to how frequently they are used, and how many registers are available. If you are optimizing for speed, the compiler gives highest priority to variables used in loops.

The compiler also gives priority to variables declared as `register`, but does not automatically assign them to registers. For example, if the compiler is choosing between a variable from an inner loop and a variable declared as `register`, it is the variable from the inner loop that the compiler places in the available register.

To prevent the compiler from reusing registers, use the `no_register_coloring` pragma directive. Refer to the *C Compilers Reference* manual for details.

See also, K&R, A4.1, A8.1.

Floating Point Support

Palm OS 2.0 and later versions include a version of the CodeWarrior 32-bit floating point library that allows you to use the standard C syntax in [Listing 7.11](#).

Listing 7.11 Floating point syntax

```
float a,b,c;  
c = a + b;
```

The built-in floating point support is limited to basic arithmetic operations. To use floating point functions such as exponentials, you must either implement them yourself or use a third party math library such as Mathlib.

The easiest way to include Mathlib in your project is to use the Palm OS Application wizard. Enable the Mathlib option in the second page of the wizard.

The Mathlib project is located here (where *CWFolder* is the CodeWarrior installation folder):

```
CWFolder\CW for Palm OS Support\Other SDKs\MathLib
```

Pragma Directives

This section describes pragma directives that are Palm OS-specific. For a complete list and detailed information about pragma directives, refer to the *C Compilers Reference* manual.

warn_a5_access

This directive causes the compiler to generate a warning message if the compiler generates code that uses the A5 register to access global variables. This pragma directive must be used outside of function definitions.

```
#pragma warn_a5_access on | off | reset
```

Use these warning messages to isolate functions that use the A5 register to access global variables so that you do not accidentally include them in shared libraries or applications that use non-global launch codes.

NOTE	The compiler cannot detect all types of code that use the A5 register to access global variables. For instance, the compiler cannot detect if code uses a jump table to call code in another segment.
-------------	---

warn_stack_usage

This directive causes the compiler to generate a warning message if a function uses more than the user-specified amount of stack space for local variables. This pragma directive must be used outside of function definitions.

```
#pragma warn_stack_usage size | off | reset
```

You can use these warning messages to identify functions that can overflow the limited stack space of Palm OS devices.

TIP	You can specify a small value (such as 1) for <i>size</i> to obtain a listing of all functions and how much stack space they use.
------------	---

C++ Issues for Palm OS

The following C++ language constructs require you to set up a global data environment (for example, A4- or A5-relative) before using them:

- polymorph (virtual member functions) class object construction/deconstruction
- exception handling (throw)
- RTTI (dynamic_cast)
- pointer-to-member NULL compares

Most C++ standard library functions, classes, and templates do not work without global data, including `new` and `delete`. Both `new` and `delete` throw exceptions that require global variables.

NOTE	To use <code>new</code> and <code>delete</code> in the absence of global variables, you must provide your own implementations that do not throw exceptions.
-------------	---

C++ support for the Palm OS includes both global and stack based objects, namespaces, RTTI, and templates.

If an application is launched with any command other than `sysAppLaunchCmdNormalLaunch`, your application may not use any global data. This restriction also includes C++ global and static data. The reason is that the global data, referenced via the A5 register, is not set up unless the application starts with the `sysAppLaunchCmdNormalLaunch` command.

See the *C Compilers Reference* for a complete description of the CodeWarrior implementation of C++.

For information on the project settings to use for applications that use C++, see [“Target Settings Reference” on page 119](#).

For information on the libraries used to develop Palm OS applications, see [“Working With Libraries” on page 89](#).

Expanded Global Space Mode

Expanded mode applications help reduce the heap usage of your application by reducing the amount of data that is placed into the global space.

Reducing Global Space Usage

Expanded mode allocates a second chunk of memory called the expanded global space. Unlike regular global space, this memory is not allocated on the heap. Instead, it is allocated out of the database RAM where programs and data files are stored.

The expanded global space can store data such as C++ virtual tables, multi-segment jump tables, exception handling tables, and RTTI information.

To access information within the expanded global space, we use the A4 register as a pointer to the middle of this space. Although we are still limited to 64KB of data, the expanded global space is completely separate from the A5-referenced global data.

Callback Thunks

Palm OS never modifies the A5 register, the pointer to the global space. However, it makes no such guarantees for the A4 register. Palm OS only guarantees that the A4 register will be restored to its former value on return from an operating system function.

Unfortunately, there are some operating system functions that do not return to your code immediately. Instead, they call your code through a function pointer. If you pass a function pointer to the Palm OS while in expanded mode, the A4 register may not be set correctly when the operating system calls your callback function.

Thus, using expanded mode requires one modification to your program code: you must use callback thunks to preserve the A4 register.

Callback thunks are small functions that your application generates during runtime. A callback thunk restores the A4 register to its correct value, calls your function, then changes A4 back to the value the caller expects it to be when your callback returns.

To make your own thunks, include the header file `<CWCallbackThunks.h>` in your source file. Declare your thunk using the type `_CW_CallbackThunk`, then generate it at runtime using a call to `__GenerateCallbackThunk`. When you call on an operating system routine that requires a function pointer, cast the address of the thunk to the type of the function pointer that the operating system routine requires.

NOTE	Because Palm OS can call event handlers while thunks are active, you should use a call to <code>__GenerateEventThunk</code> to generate a thunk that you intend to pass to <code>FrmSetEventHandler</code> .
-------------	--

This example uses a thunk to call the `SysQSort` routine:

```
_CW_CallbackThunk sortThunk;  
__GenerateThunk(mySortRoutine, &sortThunk);  
SysQSort(&intArray, numElements, sizeof(Int64),  
(CmpFuncPtr)&sortThunk, 0);
```


Debugging

This chapter covers only those aspects of debugging that are specific to the Palm OS platform. Please see the *IDE User's Guide* for general information about the debugger and debugger panels.

The topics in this chapter are:

- [Debugger Limitations](#) — describes limitations of the CodeWarrior debugger for Palm OS development
- [Debugging Palm OS® Projects](#) — shows how to debug projects on Palm OS devices, the Palm OS Emulator, and the Palm OS Simulator
- [Stepping into Palm OS Traps](#) — describes how to configure CodeWarrior projects to allow you to use the debugger to step into Palm OS traps while debugging
- [Debugging Shared Libraries and Code Resources](#) — shows you how to debug Palm OS code resources and shared libraries using the CodeWarrior debugger
- [Using the Palm OS® Debug Console](#) — shows how to obtain access to the Palm OS debug console from within the CodeWarrior IDE

Debugger Limitations

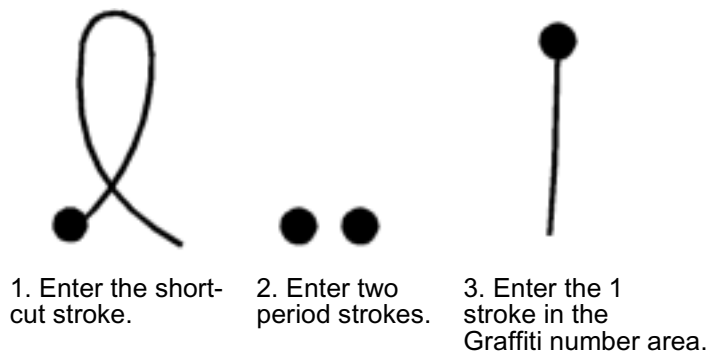
This section discusses debugger features that are not implemented for Palm OS development. These features are:

- [Stop command](#)
- [Watchpoints](#)

Stop command

The CodeWarrior debugger **Stop** command cannot halt execution of a Palm OS application. To stop a running application and send it back into debug mode, use the Graffiti strokes shown in [Figure 8.1](#) on the Palm OS Emulator, Palm OS Simulator, or Palm OS device on which the application is running.

Figure 8.1 Graffiti strokes for debug mode



Watchpoints

The CodeWarrior debugger supports only one watchpoint if you are using Palm OS Emulator version 3.3 or later. You may not use watchpoints with any other debug targets.

Debugging Palm OS® Projects

This section describes various ways to debug Palm OS projects using the IDE:

- [Debugging Using the Palm OS Emulator](#) — shows you how to configure the IDE to use the Palm OS Emulator to debug Palm OS projects
- [Debugging Using the Palm OS Simulator](#) — shows you how to configure the IDE and the Palm OS Simulator to debug Palm OS projects
- [Debugging Using a Palm OS Device](#) — shows you how to configure the IDE to use a Palm device to debug Palm OS projects

Debugging Using the Palm OS Emulator

This section describes how to configure the CodeWarrior IDE to debug Palm OS projects using the Palm OS Emulator (POSE). With POSE, you can test your application with:

- virtual devices older or newer than what you own
- foreign language ROMs
- debug-enabled ROMs that provide extra error checking and debugging features

Choosing a ROM Image

To use the emulator, you must provide it with a Palm OS ROM image. ROM images correspond to particular device configurations and Palm OS operating system versions. You can obtain ROM images either by downloading them from the PalmSource web site or by transferring them from actual Palm devices. For information on how to download Palm OS ROM image files from the PalmSource web site, see:

<http://www.palmos.com/dev/tools/emulator/#roms>

To obtain a ROM image from actual Palm device, you can select **Transfer ROM** from the emulator contextual menu. See the Palm OS Emulator documentation for more information.

NOTE	PalmSource recommends that you always download the latest ROM images from one of the following pages of the PalmSource web site: http://www.palmos.com/dev/programs/pdp/login.html http://www.palmos.com/dev/tech/hardware/
-------------	--

Loading a ROM Image

In this section, you load a ROM image into the Palm OS Emulator.

NOTE You need to locate a Palm OS Emulator program for this tutorial. The CodeWarrior Development Tools for Palm OS package includes several versions of the Palm OS Emulator located in the following folder (where *CWFolder* is your CodeWarrior installation folder):

`CWFolder\CW for Palm OS Tools\`

The Palm OS Emulator behaves differently when you run it for the very first time on a computer.

If this is the first time you have run the emulator on your computer, proceed to [“Running the Emulator for the First Time”](#), below.

If you have run the emulator on your computer before, proceed to [“Running the Emulator After the First Time” on page 207](#).

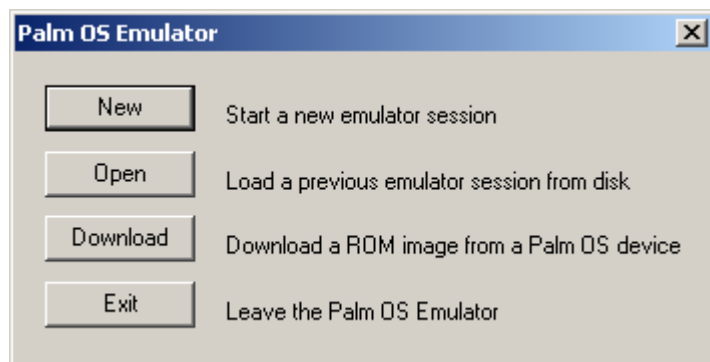
Running the Emulator for the First Time

To run the emulator for the first time:

1. Run the Palm OS Emulator.

The emulator starts up and displays the **Palm OS Emulator** startup options dialog box ([Figure 8.3](#)).

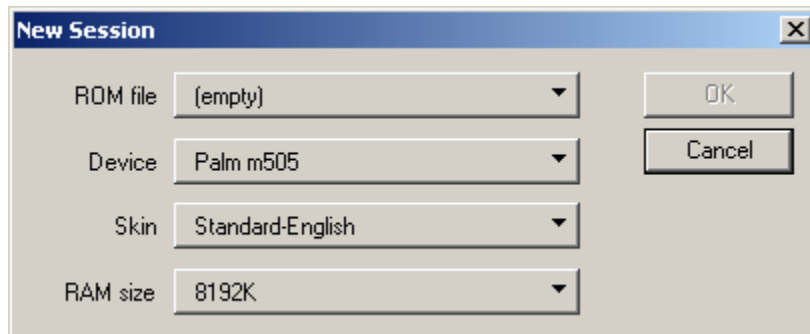
Figure 8.2 Palm OS Emulator startup options dialog box



2. Click **New**.

The emulator displays the **New Session** dialog box ([Figure 8.3](#)).

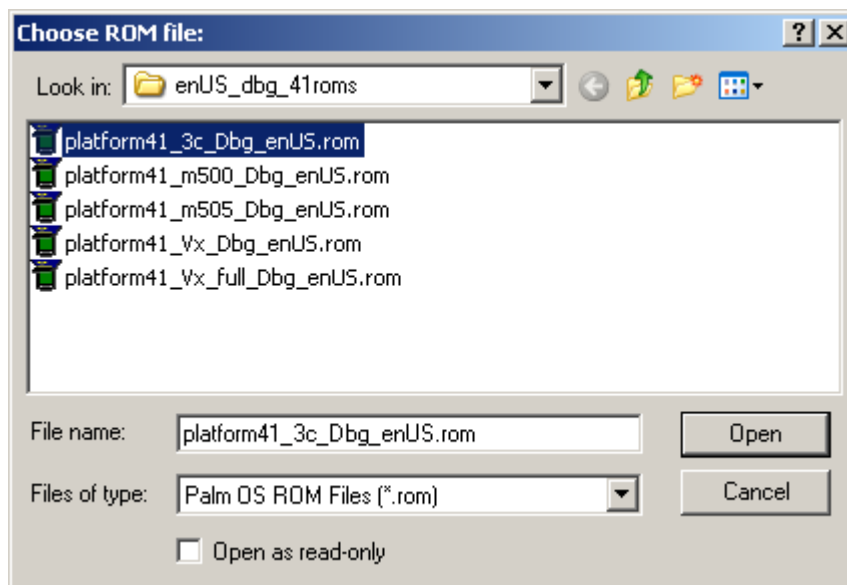
Figure 8.3 New Session dialog box



3. Select **Other** from the **ROM file** menu.

The emulator displays the **Choose ROM File** dialog box ([Figure 8.4](#)).

Figure 8.4 Choose ROM File dialog box



4. Use the **Choose ROM File** dialog box to select the ROM image file you want to use.

5. Click **Open**.

The emulator closes the **Choose ROM File** window and updates the **New Session** dialog box to show your ROM image selection.

6. Click **OK**.

The emulator closes the **New Session** dialog box, loads the ROM image file, displays a new emulator window, and runs the ROM image ([Figure 8.5](#)).

Figure 8.5 Palm OS Emulator window



You have loaded a ROM image into the emulator. Next you must configure the CodeWarrior IDE to use the emulator when you start a debug session. Proceed to [“Configuring the CodeWarrior IDE” on page 210](#).

Running the Emulator After the First Time

On subsequent runs, the emulator automatically loads the last-used ROM image file. This section assumes that the last-used ROM image file is not the ROM image file you want to use.

To run the emulator after the first time:

1. Run the Palm OS Emulator.

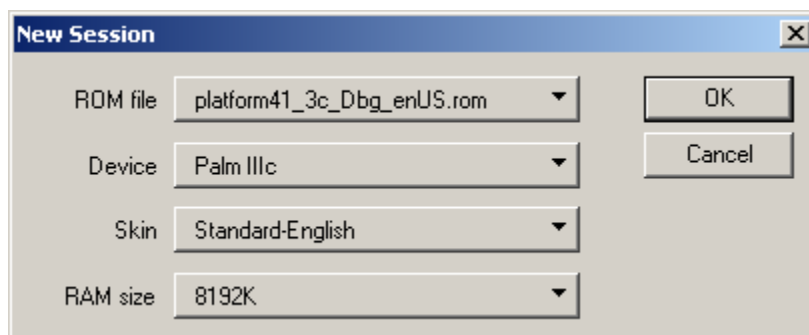
The emulator closes the **New Session** dialog box, loads the last-used ROM image file, displays a new emulator window, and runs the ROM image ([Figure 8.6](#)).

Figure 8.6 Palm OS Emulator window



2. Right-click the emulator window and select **New** from the contextual menu. The emulator displays the **New Session** dialog box ([Figure 8.3](#)).

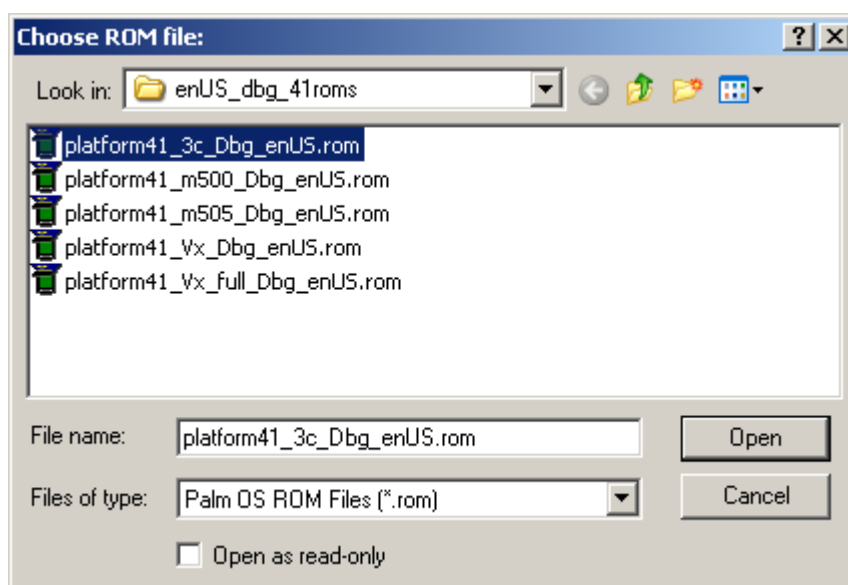
Figure 8.7 New Session dialog box



3. Select **Other** from the **ROM file** menu.

The emulator displays the **Choose ROM File** dialog box ([Figure 8.4](#))

Figure 8.8 Choose ROM File dialog box



4. Use the **Choose ROM File** dialog box to select the ROM image file you want to use.
5. Click **Open**.

The emulator closes the **Choose ROM File** window and updates the **New Session** dialog box to show your ROM image selection.

6. Click **OK**.

The emulator closes the **New Session** dialog box, loads the ROM image file, displays a new emulator window, and runs the ROM image ([Figure 8.5](#)).

Figure 8.9 Palm OS Emulator window



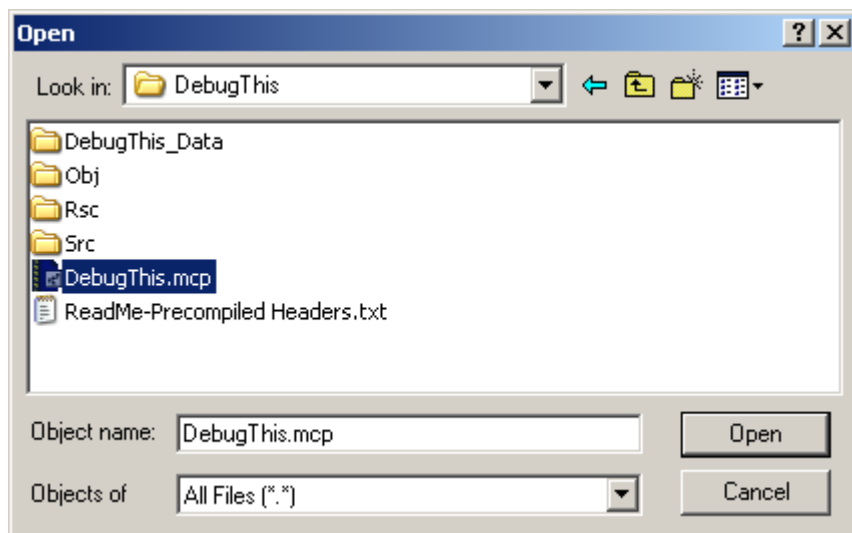
Configuring the CodeWarrior IDE

In this section, you configure the CodeWarrior IDE to connect to the Palm OS Emulator when you initiate a debug session with the IDE.

1. Start the CodeWarrior IDE.
2. Select **File > Open** from the CodeWarrior menu bar.

The IDE displays the **Open** dialog box ([Figure 8.19](#)).

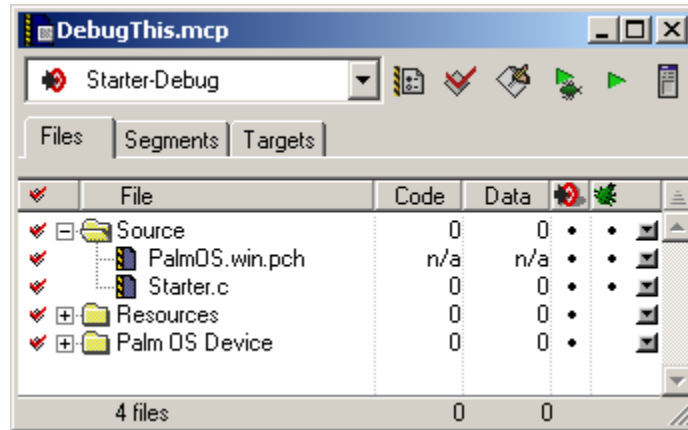
Figure 8.10 Open dialog box



3. Open the project you want to debug.

The IDE displays the Project window ([Figure 8.20](#)).

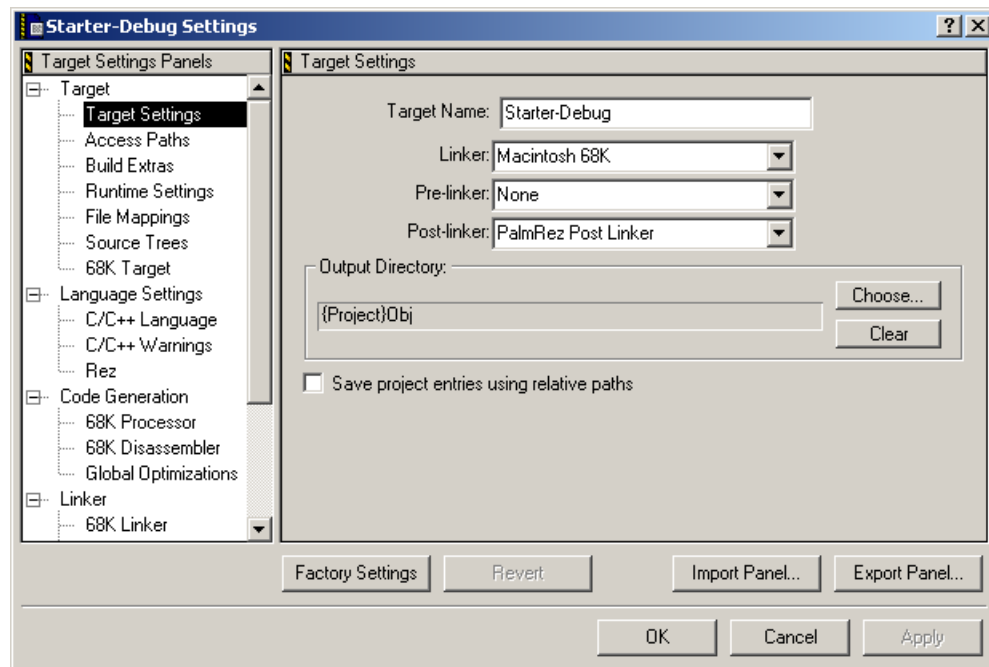
Figure 8.11 Project window



4. Select **Edit > TargetName Settings** from the CodeWarrior IDE menu bar (where *TargetName* is the name of the current build target).

The IDE displays the **Target Settings** window ([Figure 8.12](#))

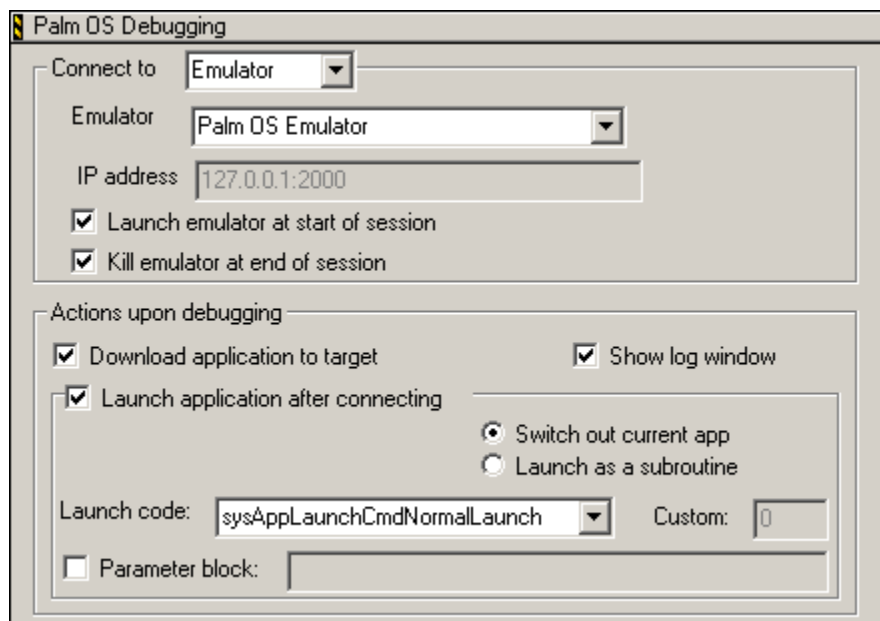
Figure 8.12 Target Settings window



5. Select **Palm OS Debugging** from the **Target Settings Panels** list on the left side of the window.

The IDE displays the **Palm OS Debugging** settings panel ([Figure 8.22](#)).

Figure 8.13 Palm OS Debugging settings panel



6. Select **Emulator** from the **Connect to** menu.
7. Select *EmulatorName* from the **Emulator** menu (where *EmulatorName* is the name of the POSE program you want the IDE to use when you debug the build target).
8. Check the **Download application to target** box.
9. Check the **Launch application after connecting** box.
10. Select the **Switch out current app** option.
11. Select **sysAppLaunchCmdNormalLaunch** from the **Launch code** menu.
12. Click **OK**.

The IDE saves your changes and closes the **Target Settings** window.

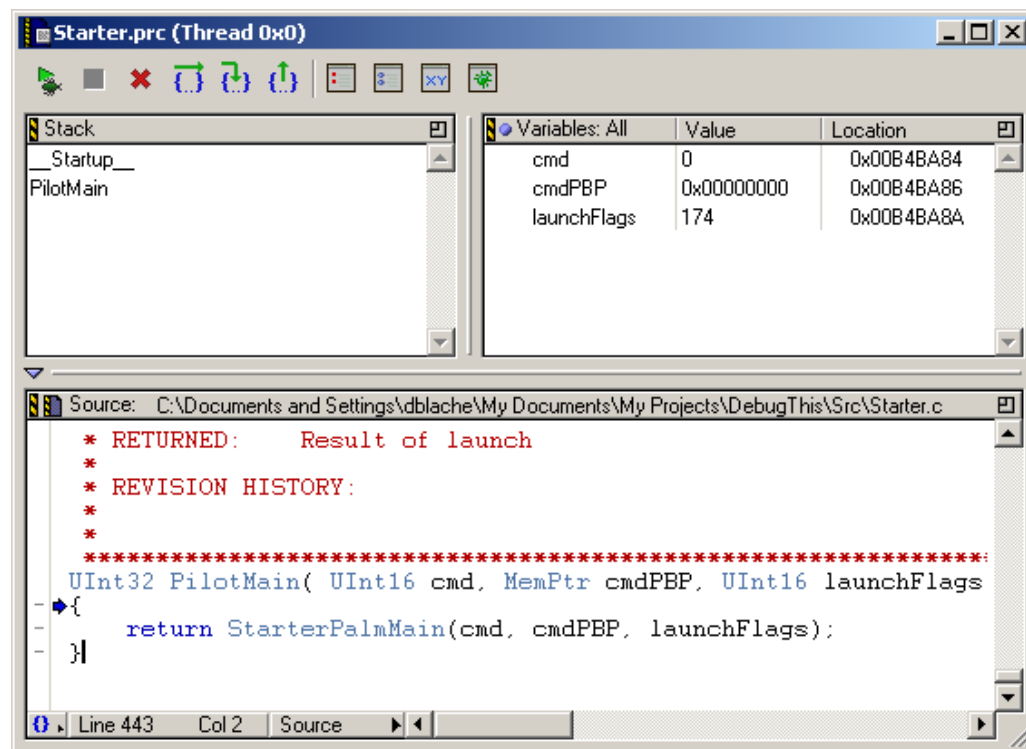
Starting the CodeWarrior Debugger

The CodeWarrior IDE should still be running from the previous section. All you need to do now is start a debug session in the IDE.

Select **Project > Debug** from the CodeWarrior menu bar.

The IDE starts a new debug session, starts the emulator if it is not running, downloads the application to the emulator, starts the application, and displays the **Thread** window ([Figure 8.14](#)).

Figure 8.14 Thread window



You can now debug the application. You can use the **Thread** window to start, stop, and step through your code. You can interact with the application using the emulator.

See the *IDE User's Guide* for complete information on using the CodeWarrior debugger.

NOTE You do not need to enter the . . 2 shortcuts into the emulator when you debug using the POSE.

Debugging Using the Palm OS Simulator

This section describes how to configure the Palm OS Simulator and the CodeWarrior IDE to debug 68K build targets using the simulator, and how to start a CodeWarrior debug session with the simulator.

Configuring the Simulator

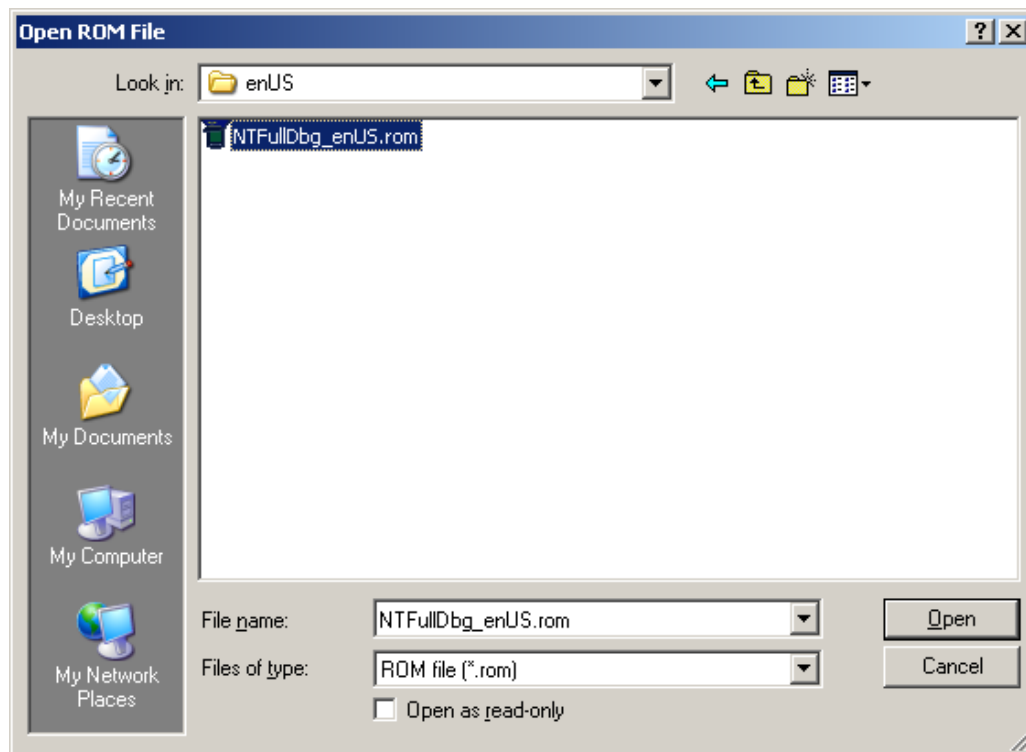
In this section, you configure the Palm OS Simulator to use an external debugger by instructing the simulator to listen to a TCP/IP port for connection attempts from the CodeWarrior debugger.

1. Start the debug version of the simulator, which is located in the following folder within the CodeWarrior installation folder:

```
CWFolder\CW for Palm OS Tools\Palm OS 5  
Simulator\Debug\PalmSim.exe
```

The simulator starts up and displays the **Open ROM File** dialog box ([Figure 8.15](#)).

Figure 8.15 Open ROM File dialog box

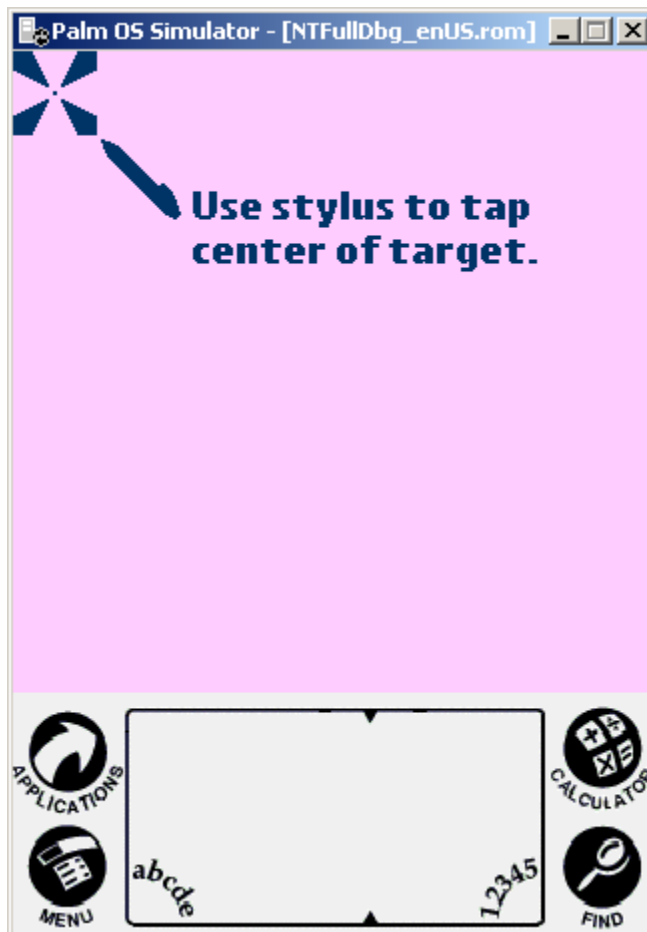


2. Select the ROM file, `NTFullDbg_enUS.rom`, which is located in the following folder within the CodeWarrior installation folder:

`CWFolder\CW for Palm OS Tools\Palm OS 5
Simulator\Debug\enUS`

The simulator starts up and displays the **Palm OS Simulator** window ([Figure 8.16](#)).

Figure 8.16 Palm OS Simulator window

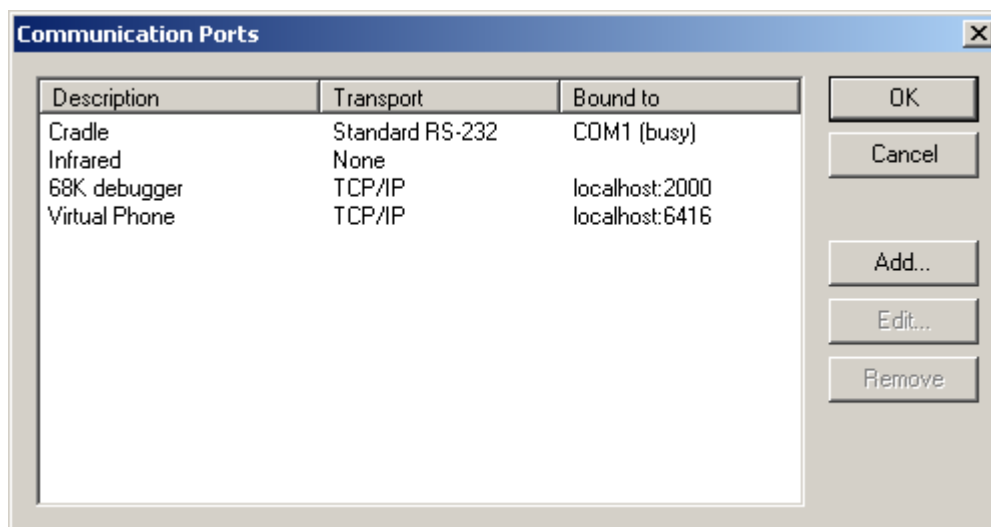


3. Right-click the **Palm OS Simulator** window to view the contextual menu and select **Settings > Communication > Redirect Net Lib Calls to Host TCP/IP**.

4. Right-click the **Palm OS Simulator** window to view the contextual menu and select **Settings > Communication > Communication ports**.

The simulator displays the **Communication Ports** dialog box ([Figure 8.17](#)).

Figure 8.17 Communication Ports dialog box



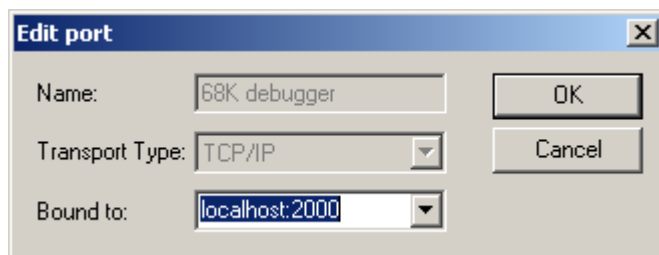
5. Select **68K debugger**.

The simulator highlights your selection.

6. Click **Edit**.

The simulator displays the **Edit Port** dialog box ([Figure 8.18](#)).

Figure 8.18 Edit Port dialog box



7. Enter "localhost:2000" in the **Bound to** text field.

8. Click **OK**.

The simulator closes the **Edit Port** dialog box.

9. Click **OK**.

The simulator closes the **Communication Ports** dialog box.

The simulator is now configured to monitor TCP/IP port 2000 to allow connections from an external debugger (in this case, the CodeWarrior debugger).

In the next section, you configure the CodeWarrior debugger to connect to the TCP/IP port the simulator monitors.

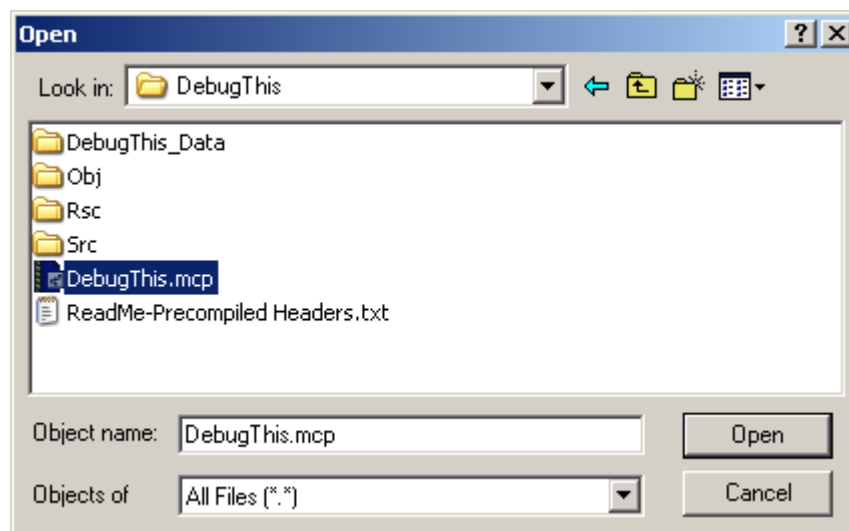
Configuring the CodeWarrior IDE

In this section, you use the CodeWarrior IDE to configure a project to connect to the Palm OS Simulator when you initiate a debug session with the IDE.

1. Start the CodeWarrior IDE.
2. Select **File > Open** from the CodeWarrior menu bar.

The IDE displays the **Open** dialog box ([Figure 8.19](#)).

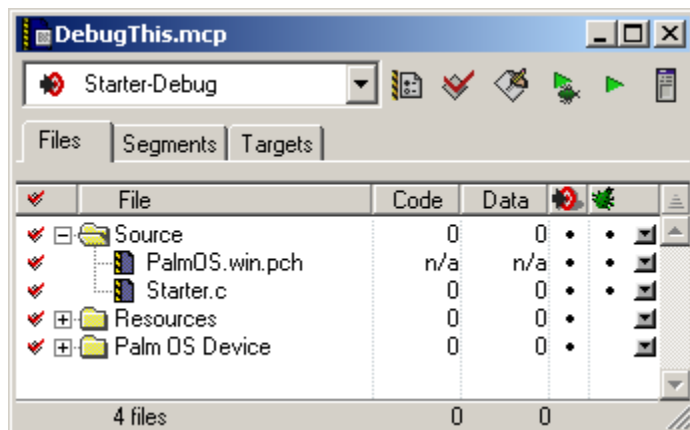
Figure 8.19 Open dialog box



3. Open the project you want to debug.

The IDE displays the Project window ([Figure 8.20](#)).

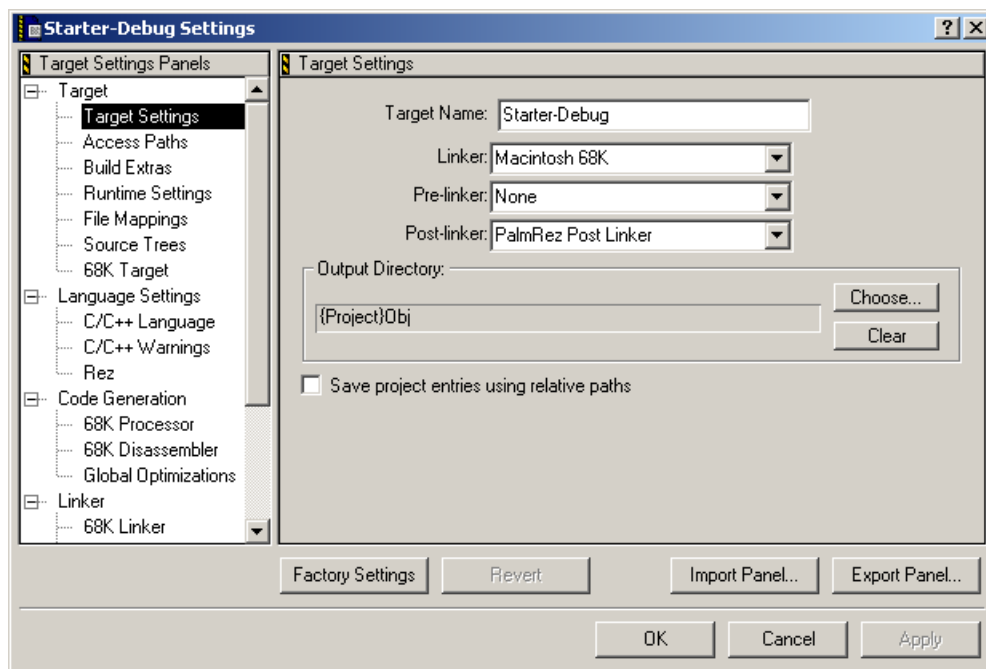
Figure 8.20 Project window



4. Select **Edit > TargetName Settings** from the CodeWarrior IDE menu bar (where *TargetName* is the name of the current build target).

The IDE displays the **Target Settings** window ([Figure 8.21](#))

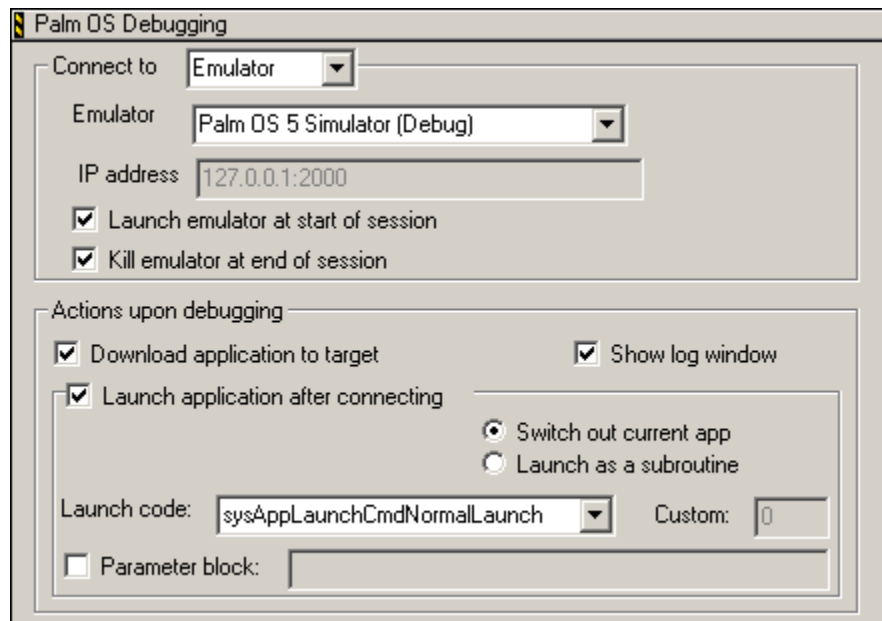
Figure 8.21 Target Settings window



5. Select **Palm OS Debugging** from the **Target Settings Panels** list on the left side of the window.

The IDE displays the **Palm OS Debugging** settings panel ([Figure 8.22](#)).

Figure 8.22 Palm OS Debugging settings panel



6. Select **Emulator** from the **Connect to** menu.
7. Select **Palm OS 5 Simulator (Debug)** from the **Emulator** menu.
8. Check the **Download application to target** box.
9. Check the **Launch application after connecting** box.
10. Select the **Switch out current app** option.
11. Select **sysAppLaunchCmdNormalLaunch** from the **Launch code** menu.
12. Click **OK**.

The IDE saves your changes and closes the **Target Settings** window.

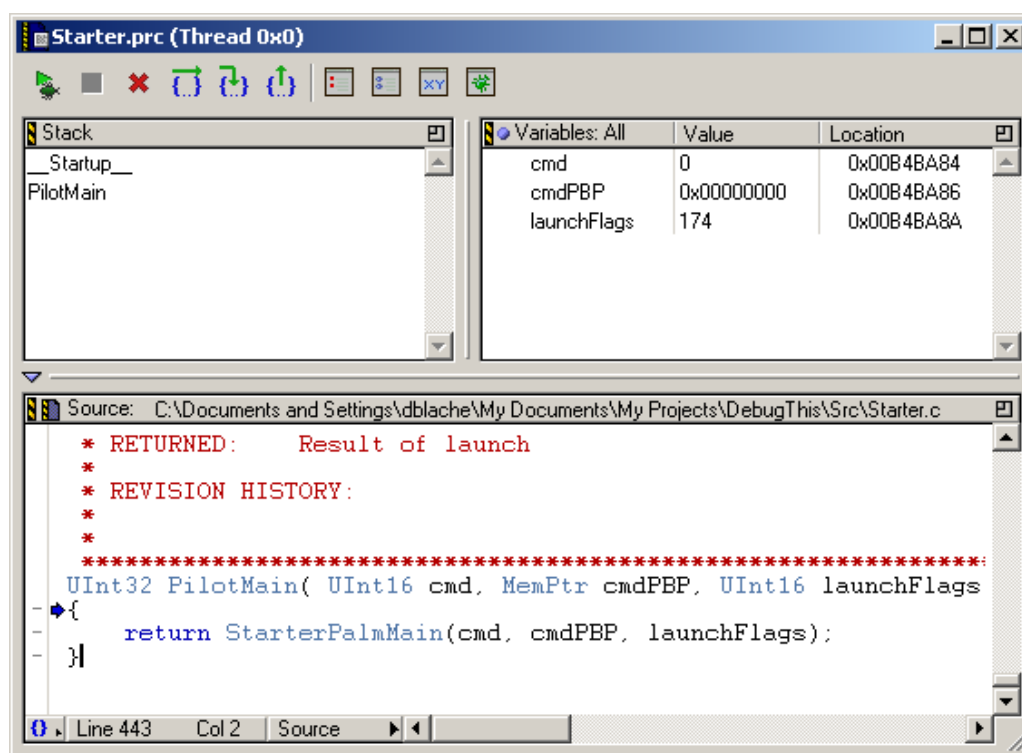
Starting the CodeWarrior Debugger

The CodeWarrior IDE should still be running from the previous section. All you need to do now is start a debug session in the IDE.

Select **Project > Debug** from the CodeWarrior menu bar.

The IDE starts a new debug session, starts the simulator if it is not running, downloads the application to the simulator, starts the application, and displays the **Thread** window ([Figure 8.23](#)).

Figure 8.23 Thread window



You can now debug the application. You can use the **Thread** window to start, stop, and step through your code. You can interact with the application using the simulator.

You can also select **PalmOS > Launch Emulator** from the CodeWarrior menu bar to manually run the simulator.

See the *IDE User's Guide* for complete information on using the CodeWarrior debugger.

Debugging Using a Palm OS Device

This section describes how to configure the CodeWarrior IDE to debug applications using a Palm OS device.

Connecting the Palm OS Device to the Computer

Connect the cradle of your Palm OS device to the serial or USB port of your computer. Plug the Palm OS device into its cradle. Remember which port you use for later.

NOTE You must close the HotSync Manager application on the host computer before you can debug on a Palm OS device on that computer. Make sure no other applications or operating system software (such as AppleTalk on Mac OS) are using the serial port to which the Palm OS device is connected.

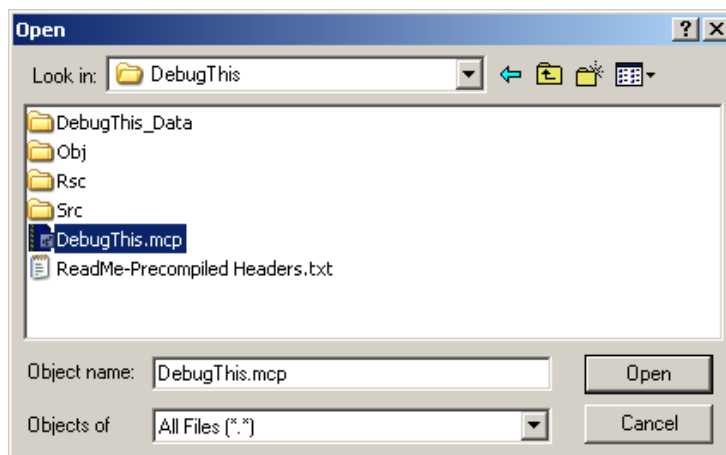
Configuring the CodeWarrior IDE

In this section, you use the CodeWarrior IDE to configure a project to connect to the Palm OS device when you initiate a debug session with the IDE.

1. Start the CodeWarrior IDE.
2. Select **File > Open** from the CodeWarrior menu bar.

The IDE displays the **Open** dialog box ([Figure 8.24](#)).

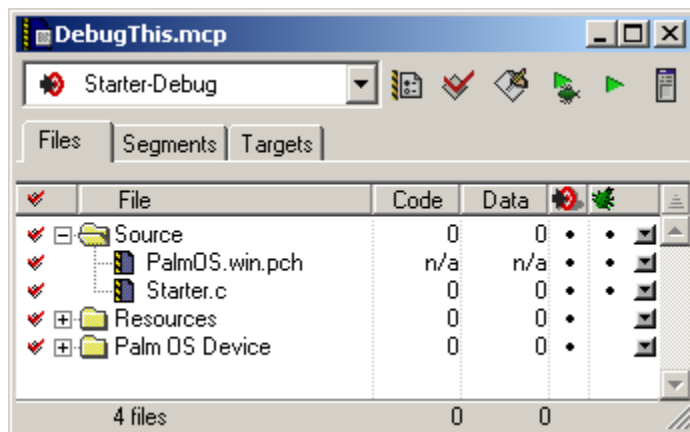
Figure 8.24 Open dialog box



3. Open the project you want to debug.

The IDE displays the Project window ([Figure 8.20](#)).

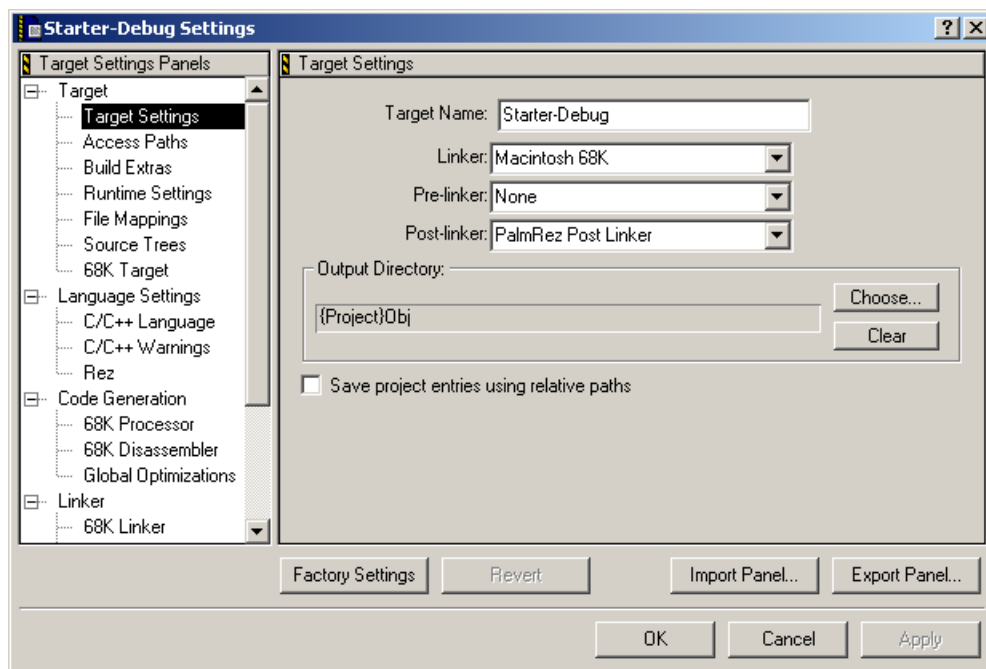
Figure 8.25 Project window



4. Select **Edit > TargetName Settings** from the CodeWarrior IDE menu bar (where *TargetName* is the name of the current build target).

The IDE displays the **Target Settings** window ([Figure 8.26](#))

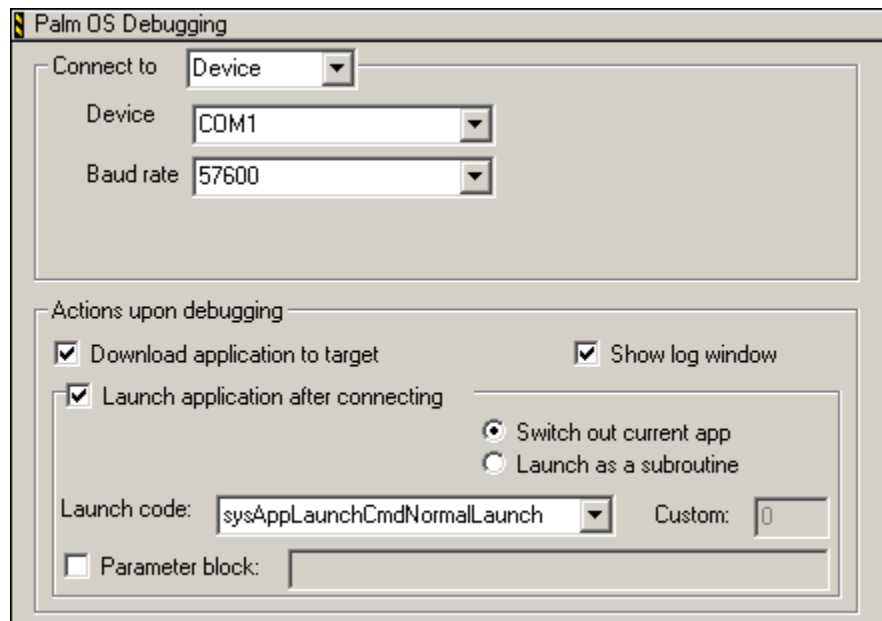
Figure 8.26 Target Settings window



5. Select **Palm OS Debugging** from the **Target Settings Panels** list on the left side of the window.

The IDE displays the **Palm OS Debugging** settings panel ([Figure 8.22](#)).

Figure 8.27 Palm OS Debugging settings panel



6. Select **Device** from the **Connect to** menu.
7. Select **PortName** from the **Device** menu (where **PortName** is the name that corresponds to the port to which you connected the device in the previous section).
8. Check the **Download application to target** box.
9. Check the **Launch application after connecting** box.
10. Select the **Switch out current app** option.
11. Select **sysAppLaunchCmdNormalLaunch** from the **Launch code** menu.
12. Click **OK**.

The IDE saves your changes and closes the **Target Settings** window.

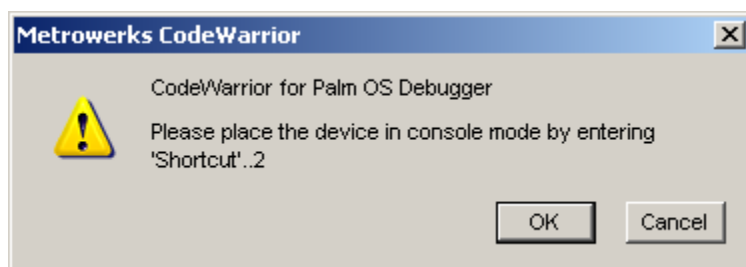
Starting the CodeWarrior Debugger

The CodeWarrior IDE should still be running from the previous section. All you need to do now is start a debug session in the IDE.

1. Select **Project > Debug** from the CodeWarrior menu bar.

The IDE displays the **Metrowerks CodeWarrior** dialog box ([Figure 8.28](#)) asking you to place the device in console mode.

Figure 8.28 Metrowerks CodeWarrior dialog box



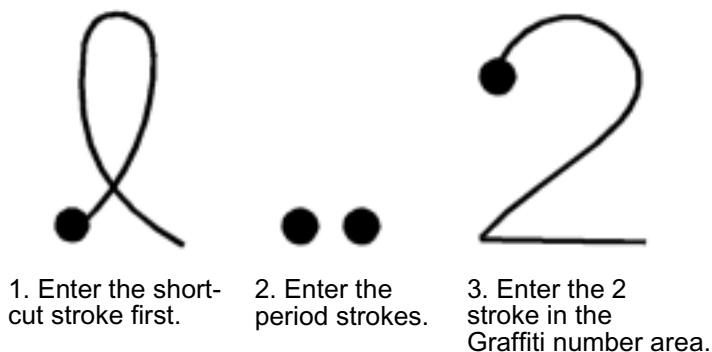
2. On the Palm OS device, tap the **Find** button.

The device displays the **Find** dialog box.

3. Enter the strokes shown in [Figure 8.29](#) to enter console mode.

If you have successfully entered these strokes and **System Sound** box is checked in the device's **Preferences** application, the device responds with a click sound.

Figure 8.29 Graffiti strokes for entering console mode



If the device you are debugging is a Handspring Visor or Prism and is in a serial cradle, you must press the **Up** button on the device before writing the 2. You must hold down the **Up** button until the CodeWarrior debugger window appears. USB cradle users may ignore this notice.

If the device you are debugging is a Handspring Visor or Treo, you may find the DebugPrefs application useful. This application lets you set the default debug mode (USB or serial) for those devices so you won't have to hold down any keys while entering debug mode. The DebugPrefs application is located here (where *CWFolder* is the CodeWarrior installation folder):

`CWFolder\CW for Palm OS Tools\Handspring Tools\DebugPrefs`

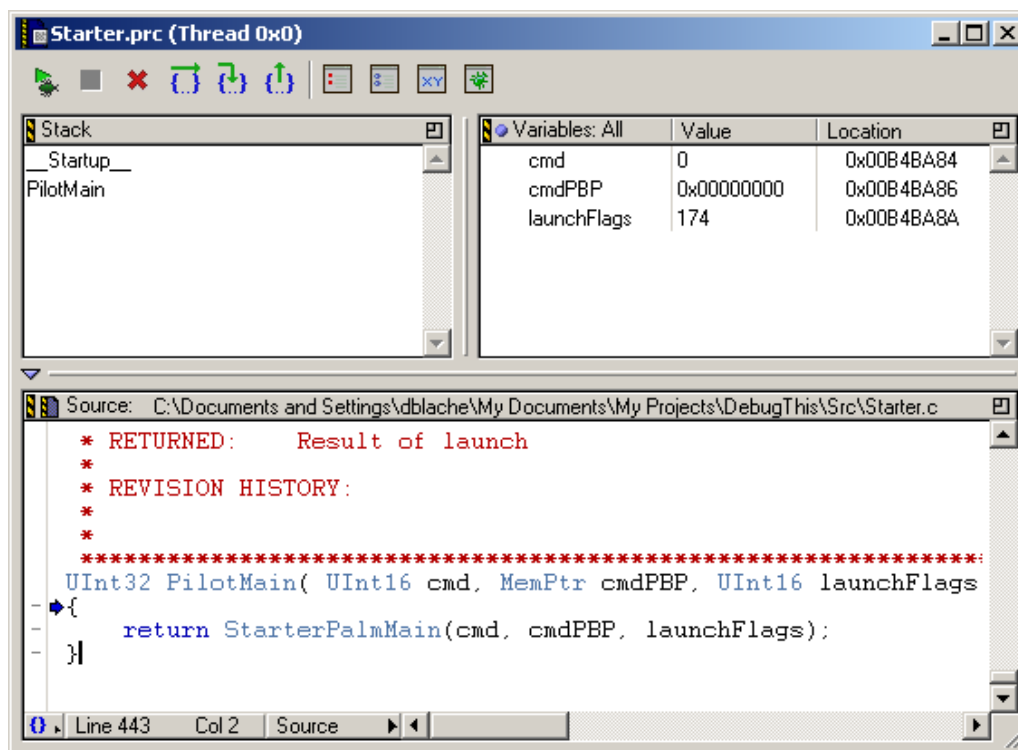
TIP	An alternative way to enter console mode is to download the DotDotTwo application and run it on your Palm OS device. The DotDotTwo project is located here (where <i>CWFolder</i> is the CodeWarrior installation folder):
------------	--

`CWFolder\ (CodeWarrior Examples)\Palm OS\Metrowerks
Examples\DotDotTwo\`

4. Click **OK** in the **Metrowerks CodeWarrior** dialog box.

The IDE starts a new debug session, downloads the application to the device, starts the application, and displays the **Thread** window ([Figure 8.30](#)).

Figure 8.30 Thread window



You can now debug the application. You can use the **Thread** window to start, stop, and step through your code. You can interact with the application using the device.

See the *IDE User's Guide* for complete information on using the CodeWarrior debugger.

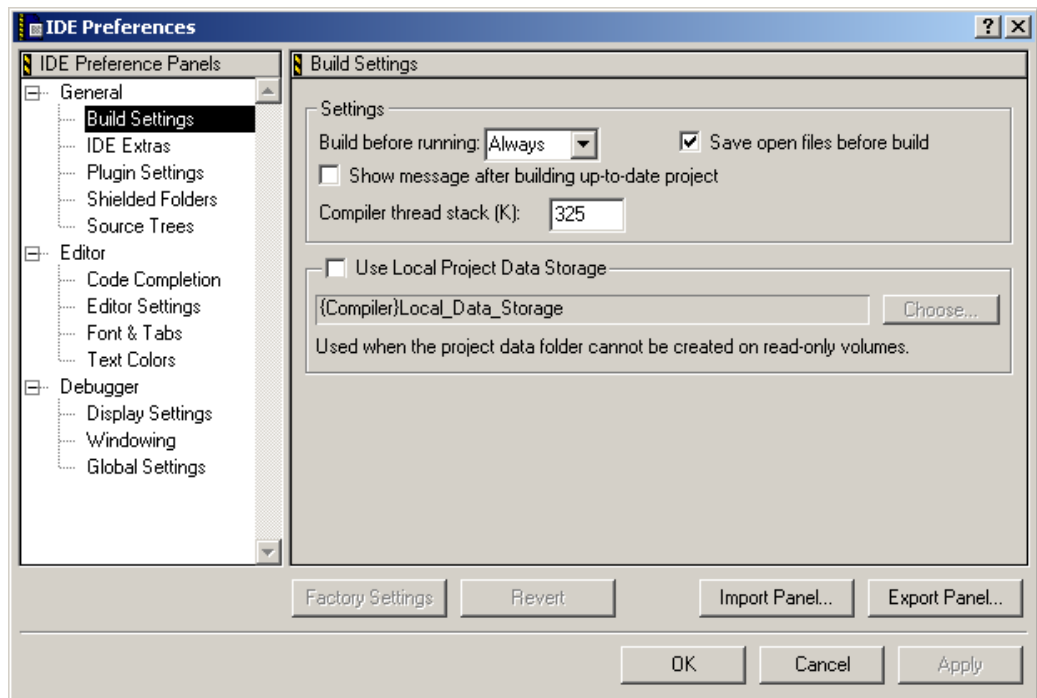
Stepping into Palm OS Traps

If you debug applications on the Palm OS Emulator, you can use the debugger to step into Palm OS traps. This section shows you how to configure the IDE to step into traps.

1. Select **Edit > Preferences** from the CodeWarrior menu bar.

The IDE displays the IDE Preferences window ([Figure 8.31](#)).

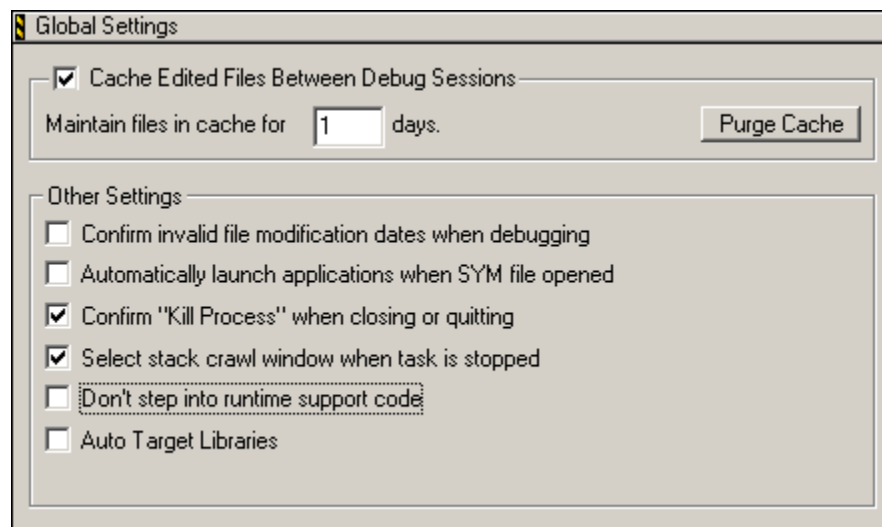
Figure 8.31 IDE Preferences window



2. Select **Global Settings** from the **IDE Preference Panels** list on the left side of the window.

The IDE displays the **Global Settings** panel ([Figure 8.32](#)).

Figure 8.32 Global Settings preference panel



3. Check the **Don't step into runtime support code** box.
4. Click **OK**.

The IDE saves your changes and closes the **IDE Preferences** window.

You can step into Palm OS traps and see the code in assembly language during subsequent debug sessions.

Debugging Shared Libraries and Code Resources

This section shows you how to debug Palm OS code resources and shared libraries using the CodeWarrior IDE.

Perform these steps sequentially:

- [Building the Host Application](#)
- [Building the Shared Library or Code Resource](#)
- [Defining the Host Application](#)
- [Starting the Debug Session](#)

Building the Host Application

First, open and build the debug version of the host application:

1. Run the CodeWarrior IDE.
2. Select **File > Open** from the CodeWarrior menu bar.
The IDE displays the **Open** dialog box.
3. Select the project file of the application that uses the shared library or code resource and click **Open**.
The IDE opens the project and displays the Project window.
4. Select **Project > Set Default Target > DebugTarget** from the CodeWarrior menu bar (where *DebugTarget* is the name of the debug build target for the application).
The IDE switches to the debug build target you specified.
5. Select **Project > Make** from the CodeWarrior menu bar.
The IDE compiles the debug build target and generates the application PRC file.
6. Select **File > Close** from the CodeWarrior menu bar to close the application project.

Ensure that the IDE built the application project without error. If there were build errors, correct them before continuing.

Building the Shared Library or Code Resource

Next, open and build the debug version of the shared library or code resource project:

1. Select **File > Open** from the CodeWarrior menu bar.
The IDE displays the **Open** dialog box.
2. Select the project file of the shared library or code resource project and click **Open**.
The IDE opens the project and displays the Project window.
3. Select **Project > Set Default Target > *DebugTarget*** from the CodeWarrior menu bar (where *DebugTarget* is the name of the debug build target for the shared library or code resource).
The IDE switches to the debug build target you specified.
4. Select **Project > Make** from the CodeWarrior menu bar.
The IDE compiles the debug build target.

Ensure that the IDE built the shared library or code resource project without error. If there were build errors, correct them before continuing.

Defining the Host Application

Next, define the host application for the shared library or code resource:

1. Select **Edit > *TargetName* Settings** from the CodeWarrior menu bar (where *TargetName* is the name of the shared library or code resource debug build target).
The IDE displays the Target Settings window.
2. Select **Runtime Settings** from the **Target Settings Panels** list.
The IDE displays the **Runtime Settings** window.
3. Click **Choose**.
The IDE displays the **Choose the Host Application** dialog box.

4. Navigate to the application PRC file that the IDE generated when you built the application project and select the PRC file.

5. Click **Open**.

The IDE closes the **Choose the Host Application** dialog box and displays the path to the application PRC file in the **Host Applications For Libraries & Code Resources** text field.

6. Click **OK**.

The IDE saves your changes and closes the **Target Settings** window.

You have configured the IDE to debug the host application and shared library or code resource.

Starting the Debug Session

Next, start a debug session with the IDE:

Select **Project > Debug** from the CodeWarrior menu bar.

The IDE starts a new debug session, downloads the application and shared library or code resource to the device, starts the application, and displays the **Thread** window.

You can now debug the application. You can use the **Thread** window to start, stop, and step through your code. You can interact with the application using the device.

See the *IDE User's Guide* for complete information on using the CodeWarrior debugger.

Using the Palm OS® Debug Console

The Palm OS debug console is a text-only, interactive console that accepts text commands, processes them, and prints text results (if applicable). You can use the debug console to examine and manipulate an application's runtime environment as well as the operating system itself during a debug session.

To access the Palm OS debug console, select **PalmOS > Open Debug Console** from the CodeWarrior menu bar. The CodeWarrior debugger attempts to establish a console session using these rules:

- If one or more debug sessions exist, the debug console connects to the device, emulator, or simulator associated with the most recently created Palm OS debug session.
- If no debug session exists, and the default project is a Palm OS project, the debug console connects to the device, emulator, or simulator associated with the default project's default target.
- If the default project is not a Palm OS project, or there is no project currently open, the debug console connects to the first emulator or simulator listed in the following XML file (where *CWFolder* is the CodeWarrior IDE installation folder):

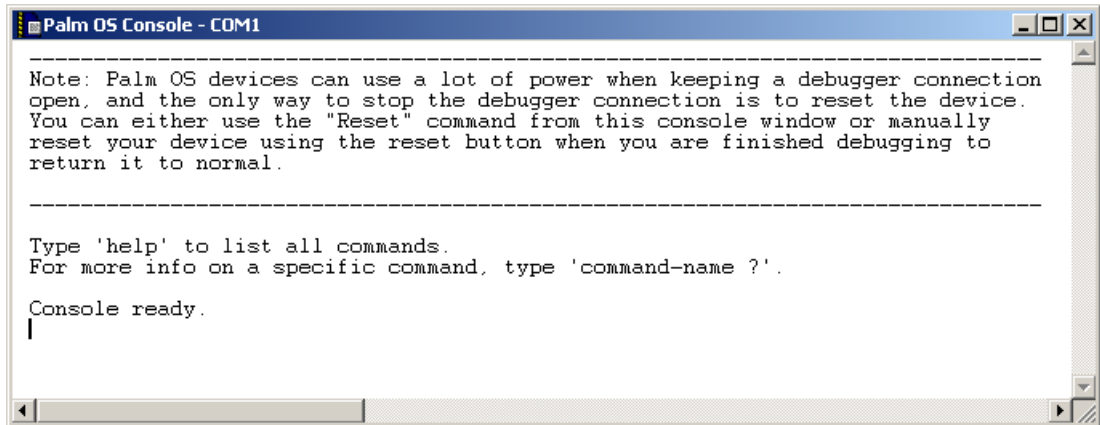
`CWFolder\Bin\Plugins\Support\Palm_OS\PalmEmulators.xml`

NOTE	The <code>PalmEmulators.xml</code> file is an XML file that describes various properties about Palm OS emulators and simulators. The IDE uses this information to populate the Emulator menu in the Palm OS Debugging target settings panel. The XML file contains detailed information about the XML tags used to describe emulators and simulators.
-------------	--

TIP	See “Debugging Palm OS® Projects” on page 203 for information on starting a Palm OS debug session with a device, emulator, or simulator.
------------	--

When the debugger establishes the console session, the IDE displays the Palm OS Console window (Figure 8.33). The console awaits your command.

Figure 8.33 Palm OS Debug Console window



Most of the commands that the console accepts are listed in the *Palm OS Development Tools Guide* in the following location (where *CWFolder* is the folder where you installed the CodeWarrior Development Tools for Palm OS tools):

```
CWFolder\CW for Palm OS Support\Documentation\Palm OS 5.0  
Docs\Development Tools Guide.pdf
```

The CodeWarrior IDE adds some additional console commands (specifically, `download` and `reset`). To obtain information about these commands, type `help` in the console.

CAUTION	The CodeWarrior debugger no longer resets the device when you close a debug console window attached to the device. Because a device's battery usage increases when a debugger connects to the device, and remains high until the device resets, we recommend that you reset the device when you are finished debugging. You can use the console <code>reset</code> command to manually reset the device.
----------------	--

Debugging

Using the Palm OS® Debug Console

Troubleshooting

This chapter provides common CodeWarrior for Palm OS problems and their solutions.

This chapter contains these sections:

- [Looking Up Palm OS Error Codes](#) — describes how to use the Palm OS Error Code search utility to look up error code information
- [Debugger Problems](#) — describes various debugger-related problems and solutions
- [Compiling and Linking Problems](#) — describes various compiling and linking problems and their solutions
- [Other Problems](#) — describes miscellaneous problems and solutions

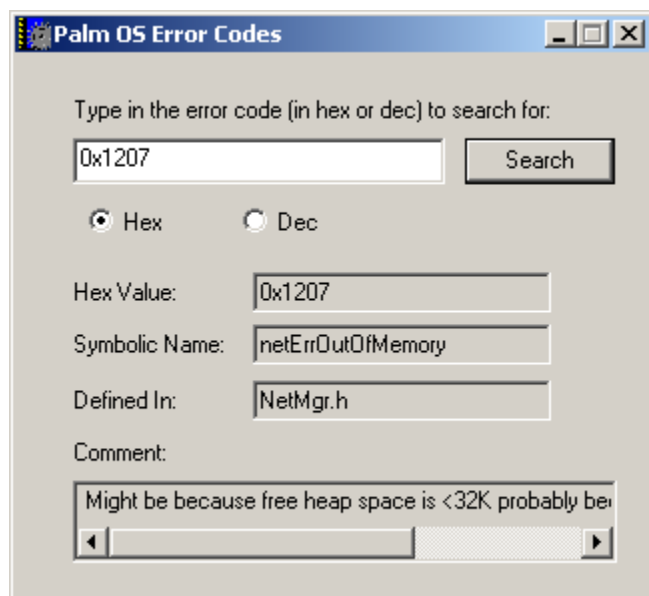
Looking Up Palm OS Error Codes

The CodeWarrior IDE has a built-in utility that allows you to look up information about Palm OS runtime error codes, such as the hex value, the symbolic name, the header file in which it is defined, and any additional comments. To access the Palm OS error code lookup utility:

1. Select **PalmOS > Search Palm OS Error Codes** from the CodeWarrior menu bar.

The IDE displays the **Palm OS Error Codes** window ([Figure A.1](#)).

Figure A.1 Palm OS Error Codes window



2. Type the error code you want to look up.
You can type the error code as decimal or hexadecimal. Hexidecimal values must start with the 0x characters.
3. Select **Hex** or **Dec** to indicate whether the error code you typed is in hexadecimal or decimal format.
4. Click **Search**.

The IDE looks up the error code you specified and updates the display to show information about the error code.

Debugger Problems

This section contains solutions to some common debugger problems:

- [Stop Command Does Not Work](#)
- [Cannot Access Serial Port Error Message](#)
- [Problems After Cancelling a Transfer](#)
- [Short Battery Life](#)
- [Device Display Flashes After Quitting or Resetting](#)

Stop Command Does Not Work

Problem

You cannot use the **Debug > Stop** command.

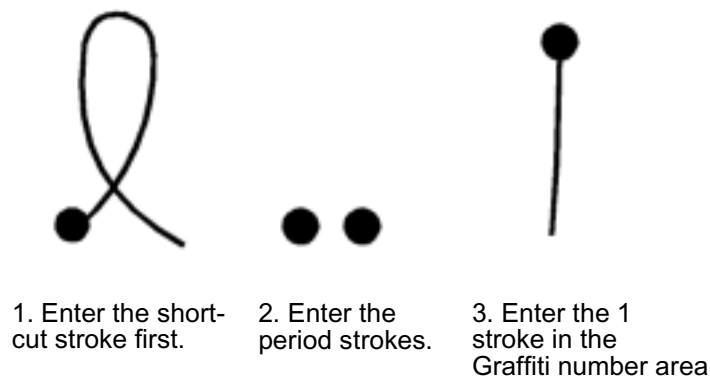
Background

The **Stop** command is currently not supported.

Solution

Enter the Graffiti strokes shown in onto the Palm OS device to force the device to go into debug mode.

Figure A.2 Graffiti strokes for debug mode



Cannot Access Serial Port Error Message

Problem

A message appears stating that the serial port cannot be accessed because it is in use.

Background

The serial port of the device was not closed during the last debug session.

Solution

Use a soft reset to close the serial port.

Problems After Cancelling a Transfer

Problem

The device crashes or behaves strangely after you cancel a data transfer operation to a device.

Background

Interrupting the CodeWarrior IDE while it uploads an application to a Palm OS device can place the device in an unknown state, causing it to function improperly.

Solution

Reset the device to correct the problem.

Short Battery Life

Problem

The batteries in the device run down quickly during or after debugging.

Background

While debugging, some of the device's power management features are disabled. This results in greater drain on battery power.

If the debugger is not able to terminate the debug session normally, the, it may not restore the power management functions.

Solution

Reset the device after your debugging session to ensure that the device restores its normal power management settings.

Device Display Flashes After Quitting or Resetting

Problem

The LCD flashes repeatedly after you quit the CodeWarrior debugger or reset the device.

Background

This problem usually occurs when a system crash prevents a proper restart of the application that you were debugging.

Solution

The solution to this problem differs depending on the version of Palm OS running on the device:

Palm OS 2.0 and Earlier

If the device is running Palm OS 2.0 or an earlier version, press and hold the up button on the device until the LCD stops flashing. This may take up to 10 seconds. Use the Memory application to then delete the application.

Palm OS 3.0 and Later

If the device is running Palm OS 3.0 or a later version, press the up button while resetting the device to do a no-notification reset. This starts Palm OS without telling any of the applications on the device that a reset happened. You can then use the application launcher to delete the application you were debugging. Once you have deleted the problem application, reset the device again to restore normal operation.

Compiling and Linking Problems

This section covers common problems you might encounter when trying to compile or link a build target. They are:

- [Projects From Earlier Releases Do Not Link](#)
- [Linker Error: 16-bit Reference is Out of Range](#)
- [Linker Error: `__RuntimeModule : Entry Point ' __InitCode '` is undefined](#)
- [Link Errors After Deleting a .tmp File \(Windows only\)](#)
- [Postlinker Error: Error Copying resource.frk](#)

Projects From Earlier Releases Do Not Link

Problem

When build a project that was created using a previous version of CodeWarrior Development Tools for Palm OS, you receive linker error messages about symbols that are referenced, but undefined.

Background

Some runtime and support libraries for Palm OS have new names. Projects that you created using an earlier releases of CodeWarrior Development Tools for Palm OS might still refer to the obsolete names of these libraries.

Solution

In your project, remove and add the libraries outlined in [“Converting Older Projects to Version 9” on page 40](#). For more information on runtime and support libraries, see [“Runtime Libraries” on page 98](#).

Linker Error: 16-bit Reference is Out of Range

Problem

The linker gives a “16-bit reference is out of range” error message

Background

Applications on Palm OSs must use 16-bit addressing. This limits jumps to a range of +/-32K bytes, which in turn limits the size of a small application to 64K bytes.

Solution

Repartition your application according to the addressing suggestions given in [“Single-Segment Application Limitations” on page 55](#) and [“Converting Single-Segment Applications to Multi-Segment Applications” on page 57](#).

Linker Error: __RuntimeModule__: Entry Point '__InitCode__' is undefined

Problem

The linker gives this message:

```
Link Error: __RuntimeModule__: Entry Point '__InitCode__' is undefined
```

Background

You must add the appropriate Palm OS Runtime library to your project if:

- you add C++ source code files to a basic C project that uses `StartupCode.lib`.
- you enable the Force C++ Compilation compiler settings option.

Solution

Add a Palm OS Runtime library to your project. See [“Runtime Libraries” on page 98](#).

Link Errors After Deleting a .tmp File (Windows only)

Problem

After you delete a temporary (`.tmp`) file in the same folder as a Palm OS project, rebuilding the same project results in link error messages that contain high-ASCII characters. This problem occurs only if you rebuild the project in Windows.

Background

The CodeWarrior IDE uses a temporary file to link projects into Palm OS applications. When it creates a `.tmp` file, it also looks in the `Resource.frk` folder, which is in the same folder as the project, for a corresponding `.tmp` file. If a `.tmp` file already exists, the IDE uses its contents.

Solution

If you delete a `.tmp` file from your project folder, make sure you also delete its corresponding file in the `Resource.frk` folder.

Postlinker Error: Error Copying resource.frk

Problem

The post-linker gives an error message while copying or creating `resource.frk`.

Background

Resources created with Constructor must have a data fork and a resource fork. In Windows, it looks as if you have a 0-byte `.rsrc` file, and then a `Resource.frk` folder in the same folder that contains a non-zero-byte `.rsrc` file of the same name.

Solution

Ensure that you have this folder hierarchy and the two copies of the `.rsrc` file, then recompile.

Other Problems

This section lists miscellaneous problems:

Application Crashes When Launched

Problem

An application crashes the device immediately when launched.

Background

There are many possible causes for an application crashing on a device. The most common explanation is missing or displaced runtime libraries in your project's build target. The compiler and linker generate object code that depends on these runtime libraries to set up the runtime environment for the application.

If you created your project without using wizards or stationery, the project may be missing libraries.

Solution

Make sure your project uses the same libraries used in the stationery project for Palm OS applications. Also, see [“Runtime Libraries” on page 98](#). If this does not work, please refer to web resources or developer support (see [“Where to Go from Here” on page 12](#)).

Index

Symbols

.MAP extension 167
.tmp filename extension 244
.xMAP extension 152

Numerics

16-bit reference out of range 55
4-byte 'int'
 68K Processor settings panel 160
4-Byte Ints option, 68K Processor panel 190
68K Disassembler settings panel 164
 Disassemble Exception Tables 164
 Only Show Operands and Mnemonics 164
 Show Code 164
 Show Data 164
 Show Name Table 165
 Show Source Code 164
 Show SYM Debugging Information 165
68K Linker settings panel 166
 Dead-strip Static Initialization Code 168
 Full Path in SYM Files 166
 Generate Link Map 167
 Generate SYM File 166
 Link Single Segment 167
 Merge Compiler Glue Into Segment 1 167
 Suppress Warning Messages 167
68K number formats 190–192
68K Processor settings panel 159
 4-byte 'int' 160
 8-byte 'double' 160
 Code Model 161
 Generate MacsBug Debug Symbols 160
 PC-Relative Constant Data 163
 PC-Relative Strings 162
 Struct Alignment 159
68K register variables 194
68K Target panel
 Mac OS Application pane
 Creator 134
 File Name 134
 Minimum Heap Size 135
 Preferred Heap Size 135
 Project Type 134
 SIZE Flags 134
 Startup Code 135
 Type 135
 Mac OS Code Resource pane
 Creator 137
 Display Dialog 137
 Extended Resource 137
 File Name 136
 Header Type 137
 Merge To File 137
 Project Type 136
 ResID 137
 Resource Flags 138
 Resource Name 136
 ResType 137
 SegType 137
 Sym Name 136
 Type 137
 Palm OS Application pane
 Expanded Mode 129
 Expanded Mode with A5-based Jumptable 129
 File Name 128
 Project Type 128
 Standard Mode 128
 Palm OS Code Resource pane
 Entry Point 131
 File Name 130
 Project Type 130
 ResID 131
 Resource Name 131
 ResType 131
 Sym Name 131
 Palm OS Static Library pane
 Code Resources 133
 Expanded Mode 132
 Expanded Mode with A5-based Jumptable 133
 File Name 132
 Project Type 132
 Standard Mode 132
68K Target settings panel 127
 Mac OS Application pane 134
 Mac OS Code Resource pane 136
 Palm OS Application pane 128
 Palm OS Code Resource pane 130
 Palm OS Static Library pane 132
8-byte 'double'
 68K Processor settings panel 160

8-Byte Doubles option, 68K Processor panel 192

A

A4 register 194

A4-relative data 198

A5-relative data 197

A6 stack frames 194

Access Paths panel. *See IDE User Guide*

Alignment

Rez settings panel 157

allocating variables 194

Allow Edit IDs

PiRC Settings panel 155

Application Name

Palm OS 68K Target panel 141, 143, 145, 147, 149

Application Version

Palm OS 68K Target panel 141, 143, 145, 147, 149

applications

adding ARMlets 58–87

changing link order 55

converting single-segment to multi-segment 57

creating 45–54

expanded mode 198

jump islands 56

multi-segment 44

single-segment 44

single-segment limitations 55

smart code model 56

working with 43–87

AppResourceList 109

arguments

inline assembly 185, 186

registers, passing in 180

ARMlets

adding to applications 58–87

asm, inline assembly 182

B

Backup

PRC File Settings panel 154

Baud rate

Palm OS Debugging settings panel 175

bitfields 194

bool size 190

Build Extras panel. *See IDE User Guide*

build process 14–18

build target 13

Bundle

PRC File Settings panel 154

C

C++ init code, strings 162

C/C++ Language panel. *See C Compilers Reference*

C/C++ Warnings panel. *See C Compilers Reference*

callback thunks 198

calling conventions 180, 193

case sensitivity, inline assembly 182

char size 190

Character Set

PiRC Settings panel 156

Code Entry Point

Palm OS 68K Target panel 141, 142, 144, 146, 149

Code Model

68K Processor settings panel 161

Code Model option, 68K Processor panel 161

code resource

defined 31

Code Resource ID

Palm OS 68K Target panel 141, 142, 145, 147, 149

Code Resource Type

Palm OS 68K Target panel 141, 142, 144, 146, 149

Code Resources

68K Target panel 133

code resources

debugging 229

CodeWarrior

build target 13

IDE overview 13

project manager 13

CodeWarrior build process 14–18

CodeWarrior command-line tools

described 20

CodeWarrior development tools 19–28

CodeWarrior IDE. *See CodeWarrior Integrated*

Development Environment

CodeWarrior Integrated Development Environment 20

CodeWarrior Tutorial for Palm OS book 43

command-line compilers 20

command-line linkers 20

command-line tools

described 20

comments, inline assembly 183

Compile For Locale

- PiRC Settings panel 156
- compiler
 - inline assembly. *See* inline assembly
 - language extensions 179–180
 - Palm OS 177–199
 - pragma directives 196
 - troubleshooting 241–244
- See also C Compilers Reference*
- compilers
 - command-line 20
- Conduit Development Kit
 - described 28
- conduits 28
- Connect to
 - Palm OS Debugging settings panel 175
- console
 - debugger 232
- Constructor for Palm OS
 - described 27
- Constructor for Palm OS 27
- Convert '...' To Elipses
 - PiRC Settings panel 155
- converting projects 40
- converting resource files 113
- creating applications 45–54
- Creator
 - 68K Target panel 134, 137
 - PalmRez Post-Linker settings panel 172
 - PRC File Settings panel 154
- Custom
 - Palm OS Debugging settings panel 176

D

- Database Name
 - PalmRez Post-Linker settings panel 172
- DB Name
 - PRC File Settings panel 153
- DB Version
 - PRC File Settings panel 154
- dc, inline assembly directive 187
- Dead-strip Static Initialization Code
 - 68K Linker settings panel 168
- debugger 201–233
 - console 232
 - described 20
 - limitations 202
 - stepping into traps 227

- troubleshooting 237–240
- debugging 201–233
 - Palm OS projects 203–226
 - Palm OS Emulator 203
 - shared libraries and code resources 229
 - simulator projects 26
 - using a Palm OS Device 221–226
 - using Palm OS Emulator 203–213
 - using Palm OS Simulator 214–220
 - See also CodeWarrior IDE User's Guide*
- Development 28
- development tools 19–28
- Device
 - Palm OS Debugging settings panel 175
- Disallow Copying
 - PRC File Settings panel 154
- Disallow Resource Overlays
 - Palm OS 68K Target panel 141, 143, 145, 147, 149
- Disassemble Exception Tables
 - 68K Disassembler settings panel 164
 - ELF Disassembler settings panel 170
- Display Dialog
 - 68K Target panel 137
- double size 192
- Download application to target
 - Palm OS Debugging settings panel 176
- ds, inline assembly directive 187

E

- ELF Disassembler settings panel 169
 - Disassemble Exception Tables 170
 - Show Addresses and Object Code 170
 - Show Code Modules 170
 - Show Comments 170
 - Show Data Modules 170
 - Show Debug Info 170
 - Show Headers 169
 - Show Relocations 170
 - Show Source Code 170
 - Show Symbol and String Tables 169
 - Use Extended Mnemonics 170
 - Verbose Info 170
- Emulator (*menu*)
 - Palm OS Debugging settings panel 175
- Entry Point
 - 68K Target panel 131
- entry, inline assembly directive 187

- error messages
 - linker 244
 - lookup 236
- Escape Control Chars
 - Rez settings panel 157
- Expanded Mode
 - 68K Target panel 129, 132
- expanded mode 98, 198
 - thunks 198
- Expanded Mode with A5-based Jumptable
 - 68K Target panel 129, 133
- Extended Resource
 - 68K Target panel 137
- extern keyword 31, 90

F

- File Mappings panel. *See IDE User Guide*
- File Name
 - 68K Target panel 128, 130, 132, 134, 136
 - Palm OS 68K Target panel 141, 142, 144, 146, 149
- Filter Mode
 - Rez settings panel 157
- float size 192
- floating-point
 - formats 192
 - libraries 195
- fralloc, inline assembly directive 185, 188
- frfree, inline assembly directive 185, 188
- Full Path in SYM Files
 - 68K Linker settings panel 166

G

- Generate Link Map
 - 68K Linker settings panel 167
 - Palm OS ARMlet Target settings panel 151
- Generate MacsBug Debug Symbols
 - 68K Processor settings panel 160
- Generate SYM File
 - 68K Linker settings panel 166
- global data 100, 162, 197, 198
- global variables, inline assembly 184

H

- hack
 - defined 32
- hacks 162, 173

- resource type for 131
- Header Type
 - 68K Target panel 137
- heap 163, 198
- hexadecimal constants, inline assembly 182
- Hidden
 - PRC File Settings panel 154
- HotSync Manager 221

I

- inline assembly 181–189
 - arguments 185, 186
 - asm 182
 - asm blocks 182
 - case sensitivity 182
 - comments 183
 - directives 187–189
 - dc 187
 - ds 187
 - entry 187
 - fralloc 185, 188
 - frfree 185, 188
 - machine 188
 - naked 189
 - opword 189
 - return 189
 - function-level 182
 - functions, initializing 185
 - global variables 184
 - hexadecimal constants 182
 - instructions 181
 - labels 183
 - local variables 185, 186
 - optimization restrictions 183
 - preprocessor 184
 - returns 186
 - struct 184
 - structures 184
 - syntax 182–186
 - variables, global 184
 - variables, local 185, 186
- inline data 179
- int size 191
- integer formats 190
- intermediate representation 13
- IP Address
 - Palm OS Debugging settings panel 175
- IR 13

J

jump islands 56

K

Kill command, Debugger 202

Kill emulator at end of session

 Palm OS Debugging settings panel 175

L

labels, inline assembly 183

language extensions 179–180

Launch application after connecting

 Palm OS Debugging settings panel 176

Launch as a subroutine

 Palm OS Debugging settings panel 176

Launch code

 Palm OS Debugging settings panel 176

Launch Constructor for Palm OS menu item 27

Launch emulator at start of session

 Palm OS Debugging settings panel 175

Launch Emulator menu item 26

libraries

 floating-point 195

 MathLib 195

 MSL C 101

 MSL C++ 103

 MSL for Palm OS 101–103

 runtime 98–100

 shared 162

limitations

 debugger 202

link map 150, 151, 167

link order 55

Link Single Segment

 68K Linker settings panel 167

Linker

 Target Settings panel 124

linker

 error messages 244

 troubleshooting 241–244

linkers

 command-line 20

List Unused Objects

 Palm OS ARMlet Target settings panel 151

local variables, inline assembly 185, 186

long double size 192

long long size 191

long size 191

M

M68000 Family Programmer's Reference Manual 181

Mac OS Merge Linker 125

Mac OS Merge linker

 described 25

 using to convert resources 113

Mac Resource Files

 PalmRez Post-Linker settings panel 171

machine, inline assembly directive 188

Macintosh 68K compiler

 described 21

Macintosh 68K Linker 124

MathLib 195

Max String Width

 Rez settings panel 157

menu item

 Launch Constructor for Palm OS 27

 Launch Emulator 26

 Object Library for Palm OS Reference 27

 Search Palm OS Error Codes 236

Merge Compiler Glue Into Segment 1

 68K Linker settings panel 167

Merge To File

 68K Target panel 137

Minimum Heap Size

 68K Target panel 135

Minimum Stack Size

 Palm OS 68K Target panel 141, 143, 145, 147, 149

modifying

 target settings 119–176

MSL for Palm OS 101–103

multi-segment application

 defined 30, 44

Multi-Segment Code Resource

 Palm OS 68K Target panel 143, 145, 147, 149

Multi-segment Code Resource

 Palm OS 68K Target panel 141

N

naked, inline assembly directives 189

number formats 190–192

O

- Object Library for Palm OS
 - described 27
- Object Library for Palm OS Reference menu item 27
- Object Library for Palm OS wizard 46
- Only Show Operands and Mnemonics
 - 68K Disassembler settings panel 164
- Open Debug Console command (IDE) 232
- optimizations, inline assembly restrictions 183
- options
 - See* settings panels
- opword, inline assembly directives 189
- Other Parameters
 - PiIRC Settings panel 156
- Output Directory option, Target Settings panel 124
- Output File Name
 - Palm OS ARMlet Target settings panel 151

P

- Palm OS
 - emulator. *See* Palm OS Emulator
 - error codes 236
 - simulator. *See* Palm OS Simulator
- Palm OS 68K Linker 125
- Palm OS 68K linker
 - described 23
- Palm OS 68K Target panel
 - 68K Application (Standard) pane
 - Application Name 141
 - Application Version 141
 - Code Entry Point 141
 - Code Resource ID 141
 - Code Resource Type 141
 - Disallow Resource Overlays 141
 - File Name 141
 - Minimum Stack Size 141
 - Multi-segment Code Resource 141
 - Target Type 141
 - 68K Code Resource pane
 - Application Name 145
 - Application Version 145
 - Code Entry Point 144
 - Code Resource ID 145
 - Code Resource Type 144
 - Disallow Resource Overlays 145
 - File Name 144
 - Minimum Stack Size 145
 - Multi-Segment Code Resource 145
 - Target Type 144
 - 68K Shared Library pane
 - Application Name 143
 - Application Version 143
 - Code Entry Point 142
 - Code Resource ID 142
 - Code Resource Type 142
 - Disallow Resource Overlays 143
 - File Name 142
 - Minimum Stack Size 143
 - Multi-Segment Code Resource 143
 - Target Type 142
 - 68K Static Library (Standard) pane
 - Application Name 149
 - Application Version 149
 - Code Entry Point 149
 - Code Resource ID 149
 - Code Resource Type 149
 - Disallow Resource Overlays 149
 - File Name 149
 - Minimum Stack Size 149
 - Multi-Segment Code Resource 149
 - Target Type 149
- Merged Resource File pane
 - Application Name 147
 - Application Version 147
 - Code Entry Point 146
 - Code Resource ID 147
 - Code Resource Type 146
 - Disallow Resource Overlays 147
 - File Name 146
 - Minimum Stack Size 147
 - Multi-Segment Code Resource 147
 - Target Type 146
- Palm OS 68K Target settings panel 139
 - 68K Application pane 140
 - 68K Code Resource pane 144
 - 68K Shared Library pane 142
 - 68K Static Library (Standard) pane 148
 - Merged Resource File pane 146
- Palm OS ARM compiler
 - described 24
- Palm OS ARM linker
 - described 24
- Palm OS ARMlet linker 125
- Palm OS ARMlet Target settings panel
 - Generate Link Map 151
 - List Unused Objects 151

- Output File Name 151
- Project Type 150
- Show Transitive Closure 151
- Palm OS C and C++ application wizard 51
- Palm OS debug console 232
- Palm OS Debugging settings panel 174
 - Baud rate 175
 - Connect to 175
 - Custom 176
 - Device 175
 - Download application to target 176
 - Emulator (*menu*) 175
 - IP Address 175
 - Kill emulator at end of session 175
 - Launch application after connecting 176
 - Launch as a subroutine 176
 - Launch code 176
 - Launch emulator at start of session 175
 - Parameter block 176
 - Show log window 175
 - Switch out current app 176
- Palm OS Emulator 203
 - debugging 203–213
 - described 26
- Palm OS Simulator 26
 - debugging 214–220
 - described 26
- PalmOSRuntime_2i_A4A5.lib 98
- PalmOSRuntime_2i_A5.lib 98
- PalmOSRuntime_4i_A4A5.lib 98
- PalmOSRuntime_4i_A5.lib 98
- PalmRez post-linker
 - described 25
 - using to convert resources 113
- PalmRez Post-Linker settings panel 171
 - Creator 172
 - Database Name 172
 - Mac Resource Files 171
 - Set Backup Bit 172
 - Set Copy-Prevention Bit 172
 - Set Hidden Bit 172
 - Set Reset Bit 172
 - Show Warnings 173
 - Sort Resources By Size 172
 - Text Description Files 171
 - Transliteration 172
 - Trap IDs 173
 - Type 172
 - Version 172
- Parameter block
 - Palm OS Debugging settings panel 176
- parameter pragma 180
- PC-Relative Constant Data
 - 68K Processor settings panel 163
- PC-Relative Strings
 - 68K Processor settings panel 162
- PC-Relative Strings option, 68K Processor panel 162
- PilotMain() 99
- PilRC compiler
 - described 22
- PilRC Designer
 - described 22
- PilRC Settings panel 155
 - Allow Edit IDs 155
 - Character Set 156
 - Compile For Locale 156
 - Convert '...' To Elipses 155
 - Other Parameters 156
 - Right-To-Left Support 156
 - Strip Non-Locale-Based Resources 156
 - Target Language 156
- POL wizard *See Object Library for Palm OS wizard*
- POSE. *See* Palm OS Emulator
- Post-Linker
 - Target Settings panel 125
- post-linker
 - described 25
- pragma directives
 - overview 196
 - warn_a5_access 196
 - warn_stack_usage 196
- PRC File Settings panel 153
 - Backup 154
 - Bundle 154
 - Creator 154
 - DB Name 153
 - DB Version 154
 - Disallow Copying 154
 - Hidden 154
 - Reset On Install 154
 - Type 153
- Preferred Heap Size
 - 68K Target panel 135
- Prefix File
 - Rez settings panel 158
- Pre-Linker

- Target Settings panel 125
- preprocessor, inline assembly 184
- project manager 13
- Project Type
 - 68K Target panel 128, 130, 132, 134, 136
 - Palm OS ARMlet Target settings panel 150
- projects 29–41
 - code resource 31
 - converting 40
 - creating with stationery 36
 - hack 32
 - multi-segment application 30
 - shared library 32
 - single-segment application 30
 - static library 31
 - working with 29–41

Q

- Quit command, Debugger 202

R

- register variables for 68K 194
- registers
 - argument passing 180
 - variables 194
- Res Type option, 68K Target panel 131
- Reset On Install
 - PRC File Settings panel 154
- ResID
 - 68K Target panel 131, 137
- ResID option, 68K Target panel 131
- resource files
 - converting 113
- Resource Flags
 - 68K Target panel 138
- resource list file 109
- Resource Name
 - 68K Target panel 131, 136
- Resource.frk folder 244
- Resources
 - 'tBTN' 106
 - 'tFRM' 106
 - 'tLST' 106
 - ID 106
 - identification 106
 - numbering 106
 - type 106

- ResType
 - 68K Target panel 131, 137
- return, inline assembly directives 189
- Rez compiler
 - described 23
- Rez settings panel 157
 - Alignment 157
 - Escape Control Chars 157
 - Filter Mode 157
 - Max String Width 157
 - Prefix File 158
 - Script Language 158
 - Suppress Warnings For Redeclared Resource Types 157
- Right-To-Left Support
 - PilRC Settings panel 156
- runtime libraries 98–100
- Runtime Settings panel. *See IDE User Guide*

S

- Save Project Entries Using Relative Paths
 - Target Settings panel 126
- Script Language
 - Rez settings panel 158
- Search Palm OS Error Codes menu item 236
- searching for error codes 236
- SegType
 - 68K Target panel 137
- serial port, USB port 221
- Set Backup Bit
 - PalmRez Post-Linker settings panel 172
- Set Copy-Prevention Bit
 - PalmRez Post-Linker settings panel 172
- Set Hidden Bit
 - PalmRez Post-Linker settings panel 172
- Set Reset Bit
 - PalmRez Post-Linker settings panel 172
- settings panels
 - 68K Disassembler 164
 - 68K Linker 166
 - 68K Processor 159
 - 68K Target 127
 - Access Paths. *See IDE User Guide*
 - Build Extras. *See IDE User Guide*
 - C/C++ Language. *See C Compilers Reference*
 - C/C++ Warnings panel. *See C Compilers Reference*

- ELF Disassembler 169
- File Mappings, *See IDE User Guide*
- other documentation 120
- Palm OS 68K Target 139
- Palm OS Debugging 174
- PalmRez Post-Linker 171
- PiIRC Settings 155
- PRC File Settings 153
- Rez 157
- Runtime Settings, *See IDE User Guide*
- selecting 122
- Target Settings panel 123
- shared libraries 162
 - debugging 229
- shared library
 - defined 32
- short double size 192
- short size 190
- Show Addresses and Object Code
 - ELF Disassembler settings panel 170
- Show Code
 - 68K Disassembler settings panel 164
- Show Code Modules
 - ELF Disassembler settings panel 170
- Show Comments
 - ELF Disassembler settings panel 170
- Show Data
 - 68K Disassembler settings panel 164
- Show Data Modules
 - ELF Disassembler settings panel 170
- Show Debug Info
 - ELF Disassembler settings panel 170
- Show Headers
 - ELF Disassembler settings panel 169
- Show log window
 - Palm OS Debugging settings panel 175
- Show Name Table
 - 68K Disassembler settings panel 165
- Show Relocations
 - ELF Disassembler settings panel 170
- Show Source Code
 - 68K Disassembler settings panel 164
 - ELF Disassembler settings panel 170
- Show SYM Debugging Information
 - 68K Disassembler settings panel 165
- Show Symbol and String Tables
 - ELF Disassembler settings panel 169
- Show Transitive Closure
 - Palm OS ARMlet Target settings panel 151
- Show Warnings
 - PalmRez Post-Linker settings panel 173
- signed char size 190
- simulator, Palm OS 26
- single-segment application
 - defined 30, 44
- SIZE Flags
 - 68K Target panel 134
- smart code model 56
- Sort Resources By Size
 - PalmRez Post-Linker settings panel 172
- Standard Mode
 - 68K Target panel 128, 132
- Startup Code
 - 68K Target panel 135
- Startup Code option, 68K Target panel 135
- static library
 - defined 31
- stationery
 - creating your own 37
 - described 34
 - Palm OS application 34
 - Palm OS ARMlet 35
 - Palm OS static library 35
 - using 36
- Stop command (debugger) 237
- strings, C++ init code 162
- Strip Non-Locale-Based Resources
 - PiIRC Settings panel 156
- struct 184
- Struct Alignment
 - 68K Processor settings panel 159
- structures, inline assembly 184
- Suppress Warning Messages
 - 68K Linker settings panel 167
- Suppress Warnings For Redeclared Resource Types
 - Rez settings panel 157
- Switch out current app
 - Palm OS Debugging settings panel 176
- Sym Name
 - 68K Target panel 131, 136
- sysAppLaunchCmdNormalLaunch 99, 197

T

- Target Language
 - PiIRC Settings panel 156

- Target Name
 - Target Settings panel 124
- target settings
 - modifying 119–176
 - overview 121
 - See also* settings panels
- Target Settings panel 123
 - Linker 124
 - Post-Linker 125
 - Pre-Linker 125
 - Save Project Entries Using Relative Paths 126
 - Target Name 124
- Target Type
 - Palm OS 68K Target panel 141, 142, 144, 146, 149
- 'tBTN' resource 106
- Text Description Files
 - PalmRez Post-Linker settings panel 171
- 'tFRM' resource 106
- thunks 198
- 'tLST' resource 106
- Transliteration
 - PalmRez Post-Linker settings panel 172
- Trap IDs
 - PalmRez Post-Linker settings panel 173
- traps
 - stepping into 227
- troubleshooting 235–245
 - compiling and linking problems 241–244
 - debugger problems 237–240
- Type
 - 68K Target panel 135, 137
 - PalmRez Post-Linker settings panel 172
 - PRC File Settings panel 153

U

- unsigned char size 190
- unsigned int size 191
- unsigned long long size 191
- unsigned long size 191
- unsigned short size 190
- updating
 - projects 40
- Use Extended Mnemonics
 - ELF Disassembler settings panel 170

V

- variable allocation 194

- variables
 - inline assembly, global 184
 - inline assembly, local 185, 186
- Verbose Info
 - ELF Disassembler settings panel 170
- Version
 - PalmRez Post-Linker settings panel 172

W

- warn_a5_access, pragma directive 196
- warn_stack_usage, pragma directive 196
- warning messages
 - compiler 196
 - PalmRez 173
- watchpoints 202
- wizards
 - described 38
 - Object Library for Palm OS application 46
 - Palm OS C and C++ application 51
- working with applications 43–87
- working with projects 29–41