



Palm OS[®] Debugger Guide

Palm OS[®] Developer Suite

Written by Brian Maas and Jenny Green.

Technical assistance from Greg Clayton, Matt Fassiotto, Xu Zhang, Brian Dawbin, Steve Lemke, and Phil Shoemaker.

Copyright © 2003–2004, PalmSource, Inc. and its affiliates. All rights reserved. This technical documentation contains confidential and proprietary information of PalmSource, Inc. (“PalmSource”), and is provided to the licensee (“you”) under the terms of a Nondisclosure Agreement, Product Development Kit license, Software Development Kit license or similar agreement between you and PalmSource. You must use commercially reasonable efforts to maintain the confidentiality of this technical documentation. You may print and copy this technical documentation solely for the permitted uses specified in your agreement with PalmSource. In addition, you may make up to two (2) copies of this technical documentation for archival and backup purposes. All copies of this technical documentation remain the property of PalmSource, and you agree to return or destroy them at PalmSource’s written request. Except for the foregoing or as authorized in your agreement with PalmSource, you may not copy or distribute any part of this technical documentation in any form or by any means without express written consent from PalmSource, Inc., and you may not modify this technical documentation or make any derivative work of it (such as a translation, localization, transformation or adaptation) without express written consent from PalmSource.

PalmSource, Inc. reserves the right to revise this technical documentation from time to time, and is not obligated to notify you of any revisions.

THIS TECHNICAL DOCUMENTATION IS PROVIDED ON AN “AS IS” BASIS. NEITHER PALMSOURCE NOR ITS SUPPLIERS MAKES, AND EACH OF THEM EXPRESSLY EXCLUDES AND DISCLAIMS TO THE FULL EXTENT ALLOWED BY APPLICABLE LAW, ANY REPRESENTATIONS OR WARRANTIES REGARDING THIS TECHNICAL DOCUMENTATION, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING WITHOUT LIMITATION ANY WARRANTIES IMPLIED BY ANY COURSE OF DEALING OR COURSE OF PERFORMANCE AND ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, ACCURACY, AND SATISFACTORY QUALITY. PALMSOURCE AND ITS SUPPLIERS MAKE NO REPRESENTATIONS OR WARRANTIES THAT THIS TECHNICAL DOCUMENTATION IS FREE OF ERRORS OR IS SUITABLE FOR YOUR USE. TO THE FULL EXTENT ALLOWED BY APPLICABLE LAW, PALMSOURCE, INC. ALSO EXCLUDES FOR ITSELF AND ITS SUPPLIERS ANY LIABILITY, WHETHER BASED IN CONTRACT OR TORT (INCLUDING NEGLIGENCE), FOR DIRECT, INCIDENTAL, CONSEQUENTIAL, INDIRECT, SPECIAL, EXEMPLARY OR PUNITIVE DAMAGES OF ANY KIND ARISING OUT OF OR IN ANY WAY RELATED TO THIS TECHNICAL DOCUMENTATION, INCLUDING WITHOUT LIMITATION DAMAGES FOR LOST REVENUE OR PROFITS, LOST BUSINESS, LOST GOODWILL, LOST INFORMATION OR DATA, BUSINESS INTERRUPTION, SERVICES STOPPAGE, IMPAIRMENT OF OTHER GOODS, COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, OR OTHER FINANCIAL LOSS, EVEN IF PALMSOURCE, INC. OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR IF SUCH DAMAGES COULD HAVE BEEN REASONABLY FORESEEN.

PalmSource, Palm OS, HotSync, and certain other trademarks and logos are trademarks or registered trademarks of PalmSource, Inc. or its affiliates in the United States, France, Germany, Japan, the United Kingdom, and other countries. These marks may not be used in connection with any product or service that does not belong to PalmSource, Inc. (except as expressly permitted by a license with PalmSource, Inc.), in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits PalmSource, Inc., its licensor, its subsidiaries, or affiliates. All other product and brand names may be trademarks or registered trademarks of their respective owners.

IF THIS TECHNICAL DOCUMENTATION IS PROVIDED ON A COMPACT DISC, THE SOFTWARE AND OTHER DOCUMENTATION ON THE COMPACT DISC ARE SUBJECT TO THE LICENSE AGREEMENTS ACCOMPANYING THE SOFTWARE AND OTHER DOCUMENTATION.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>).

Palm OS Debugger Guide

Document Number 3126-002

November 10, 2004

For the latest version of this document, visit

<http://www.palmos.com/dev/support/docs/>.

PalmSource, Inc.

1240 Crossman Avenue

Sunnyvale, CA 94089

USA

www.palmsource.com

About This Document

Palm OS Debugger Guide provides conceptual, guidance, and reference information for developers who want to use Palm OS Debugger to debug Palm OS® applications and shared libraries.

What This Book Contains

This book has the following organization:

- [Chapter 1, “Introducing Palm OS Debugger,”](#) on page 1 gives a conceptual overview of Palm OS Debugger.
- [Chapter 2, “Connecting Palm OS Debugger with a Target,”](#) on page 5 describes how you connect Palm OS Debugger with debug targets.
- [Chapter 3, “Setting Palm OS Debugger Preferences,”](#) on page 21 describes how to set the debug plug-in preferences for Palm OS Debugger.
- [Chapter 4, “Running Palm OS Debugger,”](#) on page 45 describes the windows that make up the Palm OS Debugger user interface.
- [Chapter 5, “Palm OS Debugger Menu Reference,”](#) on page 79 lists the menu commands that you use with Palm OS Debugger.
- [Chapter 6, “AdnDebug Manager,”](#) on page 95 provides conceptual and reference information on the ARM-based debugger nub manager.

Additional Resources

- Documentation
PalmSource publishes its latest versions of documents for Palm OS developers at
<http://www.palmos.com/dev/support/docs/>
- Training
PalmSource and its partners host training classes for Palm OS developers. For topics and schedules, check
<http://www.palmos.com/dev/training>
- Knowledge Base
The Knowledge Base is a fast, web-based database of technical information. Search for frequently asked questions (FAQs), sample code, white papers, and the development documentation at
<http://www.palmos.com/dev/support/kb/>

Table of Contents

About This Document	iii
What This Book Contains	iii
Additional Resources	iv
 1 Introducing Palm OS Debugger	 1
What Is Palm OS Debugger?	1
How Does Palm OS Debugger Compare to Palm Debugger?	2
Prerequisites for Using Palm OS Debugger.	2
Operating System Requirements	2
Debug Targets	2
Compiler Requirements	3
 2 Connecting Palm OS Debugger with a Target	 5
Overview of Debugger Communication	6
Debugging with Palm OS Garnet Devices	7
Installing the Debugger Nub.	7
Connecting Palm OS Debugger with a Palm OS Garnet Device	8
Removing the Debugger Nub from a Tungsten T Device.	10
Debugging with Palm OS Garnet Simulator	11
Connecting Palm OS Debugger with Palm OS Garnet Simulator	11
Debugging with 68K-Based Devices.	13
Connecting Palm OS Debugger with a 68K-Based Device	13
Debugging with Palm OS Emulator	15
Connecting Palm OS Debugger with Palm OS Emulator.	16
Overview of Application Debugging	17
68K-Based Application Debugging	17
ARM Subroutine Debugging.	18
Palm OS Protein Application Debugging	19
 3 Setting Palm OS Debugger Preferences	 21
How to Set Preferences	22
Importing and Exporting Preferences	24
Importing	24

Exporting	24
Communications Preferences.	25
Serial Preferences.	25
Sockets Preferences.	26
Debugger Plug-in Preferences	27
68K Preferences	27
ARM Preferences.	28
Debugger UI Preferences	32
Fonts Preferences.	32
Session Preferences	33
Disassembler Preferences	34
68K Preferences	34
ARM Preferences.	35
Download Plug-ins Preferences.	36
Protocols Preferences	37
68K Palm OS Debug Kernel Preferences	37
ARM Palm OS 5 Debug Nub Preferences	38
RTOS Plug-ins Preferences	38
Palm OS 6.0 Preferences.	38
Runtime Helpers Preferences.	39
ARM Palm OS Preferences.	40
Symbolics Preferences.	41
DWARF 1.1 Preferences	41
DWARF 2.0 Preferences	43

4 Running Palm OS Debugger 45

Palm OS Debugger Overview	46
Getting Started	47
Modifying Preferences	47
Using the Palm OS Debugger Windows	48
Source View	48
Files View	55
Breakpoints View	58
Registers View	62
Variables View	64

Global Variables View	66
Memory View	67
Processes View.	69
Stack Trace View	70
Expressions View	70
Profiler View	73
STDIO Console	73
Debug Console	75

5 Palm OS Debugger Menu Reference 79

Palm OS Debugger Menu Reference Overview	79
File	79
Open	80
Save	80
Recent File List.	80
Exit	80
Edit	81
Undo	81
Cut	81
Copy	81
Paste	82
Find	82
Find next	82
Find previous	82
Next breakpoint	82
Previous breakpoint	82
Key Bindings	82
Preferences	83
View	84
Main Toolbar	85
Window Toolbar	85
Status Bar	85
Target	85
Connect	85
Disconnect	86

Append Symbolics	86
Remove Symbolics	86
Load Memory	86
Save Memory	87
Flash Memory	88
Control	88
Run	88
Restart	88
Stop	89
Kill	89
Step	89
Step In	89
Step Out	89
Window	89
Cascade	90
Tile	90
Arrange Icons	90
Files Window	90
Breakpoints Window	91
Registers Window	91
Variables Window	91
Global Variables Window	91
Memory Window	91
Processes Window	91
Stack Trace Window	91
Expressions Window	92
Profiler Window	92
Create Dockable Windows.	92
File Names	92
Help	92
About Palm OS Debugger	93

6 AdnDebug Manager 95

AdnDebug Manager Concepts	96
Activate the ARM Debugger Nub	96

Register with Palm OS Debugger	96
AdnDebug Manager Constants	97
AdnDebug Manager Functions and Macros	98
.	103

Index	105
--------------	------------

Introducing Palm OS Debugger

This chapter provides a conceptual overview of Palm OS Debugger. It describes:

- [“What Is Palm OS Debugger?”](#) on page 1
- [“How Does Palm OS Debugger Compare to Palm Debugger?”](#) on page 2
- [“Prerequisites for Using Palm OS Debugger”](#) on page 2

What Is Palm OS Debugger?

Palm OS Debugger is a full-function debug tool that you can use to debug your Palm OS® applications and shared libraries. Palm OS Debugger provides support for multiple symbolic file debugging.

Palm OS Debugger provides the following features:

- Full assembly, mixed, and source-level debugging for 68K applications, PACE Native Objects (PNO), and Palm OS Protein applications
- Support for shared library debugging
- Support for debugging with multiple symbolic files
- Fully customizable short cut keys
- Views for watching expressions, memory, processes, registers, and variables.
- Setting and viewing breakpoints
- Support for PRC and SYM 3.5 symbolic files
- Support for ELF/DWARF 1.1 and ELF/DWARF 2.0

How Does Palm OS Debugger Compare to Palm Debugger?

- **Palm OS Debugger** is a tool for debugging 68K applications, PACE Native Objects, and Palm OS Protein applications.

Debugging 68K-Based Applications: Palm OS Debugger supports connecting to the 68K debug nub of a 68K-based device and to the 68K debug nub built into the PACE component of an ARM-based device. Palm OS Debugger also connects to the 68K debug nub of Palm OS Emulator and Palm OS Simulator.

Debugging PACE Native Objects: Palm OS Debugger supports connecting to the ARM debug nub of an ARM-based device.

Palm OS Debugger is the tool that is described in this manual.

- **Palm Debugger** is used for debugging Palm OS 68K applications on both Mac OS and Windows 95/98/NT platforms. Palm Debugger supports 68K assembly debugging, not source level debugging.

Palm Debugger supports connecting with either the console or debugging nub of a 68K-based device or Palm OS Emulator. For information about Palm Debugger, see *Palm OS Development Tools Guide*.

Prerequisites for Using Palm OS Debugger

Palm OS Debugger requires the following hardware and software components.

Operating System Requirements

Palm OS Debugger runs on Windows 2000 and Windows XP.

Debug Targets

- You can use Palm OS Debugger to debug 68K-based Palm OS application code running on a 68K-based device, an ARM-based device, and Palm OS Emulator.

- You can use Palm OS Debugger to debug PACE Native Objects on an ARM-based device (a device running Palm OS Garnet or Palm OS Cobalt).
- You can use Palm OS Debugger to debug Palm OS Protein applications on a device running Palm OS Cobalt.

Palm OS Debugger connects with any of these debug targets over a serial or socket connection.

For detailed information, see [Chapter 2, “Connecting Palm OS Debugger with a Target,”](#) on page 5.

Compiler Requirements

Palm OS Debugger can debug code that is compiled by the CodeWarrior for Palm OS tool suites, the ARM ADS 1.2 compiler, or any toolchain that can create PRC files with a supported symbolic format. Supported symbolic formats are ELF/DWARF 1.1, ELF/DWARF 2.0, and SYM 3.3.

In general, Palm OS Debugger supports any compiler that emits DWARF (Debugging With Attribute Record Format) debugging information Version 1.1 or above. However, because DWARF Version 1 and DWARF Version 2 are two different standards, Palm OS Debugger implements support for each in a separate plug-in. Only one of these plug-ins can be loaded at any given time. For best results while debugging, you should not combine code that uses different DWARF versions.

Introducing Palm OS Debugger

Prerequisites for Using Palm OS Debugger

Connecting Palm OS Debugger with a Target

This chapter describes how to connect Palm OS Debugger with a debug target.

Overview of Debugger Communication	6
Debugging with Palm OS Garnet Devices	7
Debugging with Palm OS Garnet Simulator	11
Debugging with 68K-Based Devices	13
Debugging with Palm OS Emulator	15
Overview of Application Debugging	17

Overview of Debugger Communication

Palm OS Debugger enables debugging via a debugger plug-in through a communication plug-in connection to a debugger nub on a debug target. See [Figure 2.1](#) for a graphical representation of this communication flow.

Figure 2.1 Palm OS Debugger Communication



Palm OS Debugger supports the following *communications plug-ins*:

- Serial, used primarily to connect to a Palm OS device that supports a serial connection.
- USB, used primarily to connect to a Palm OS device that supports a USB connection.
- Sockets, used to communicate with Palm OS Emulator and Palm OS Garnet Simulator.

The *debugger nub* can be any of the following debug targets:

- an ARM debugger nub, running on an ARM-based device. (**Note:** For some devices that use Palm OS Garnet, you need to install a device-specific ARM debugger nub. For devices that run Palm OS Cobalt, the ARM debugger nub is built into the device.)
- a 68K debugger nub, built into PACE on an ARM-based device or on Palm OS Garnet Simulator. (PACE is the Palm OS Application Compatibility Environment. For more information about PACE, see *Palm OS Simulator Guide*.)
- a 68K debugger nub, built into a 68K-based device.
- a 68K debugger nub, built into Palm OS Emulator.

Debugging with Palm OS Garnet Devices

When debugging with Palm OS Garnet devices, the Palm OS Debugger's **ARM Palm OS 5 Debug Nub** plug-in communicates through the serial connection with the ARM Debugger Nub for Palm OS Garnet on the device.

NOTE: Each ARM-based device needs to have a custom-built debug nub. To use an ARM-based handheld not described here, contact your handheld manufacturer to see whether a debug nub is available for your ARM-based handheld.

The following devices have a supported ARM debugger nub (at the time this book was written).

For palmOne devices Tungsten C, Tungsten E, Tungsten T, Tungsten T2, Tungsten T3, Tungsten T5, Zire 71, and Zire 72: These debugger nubs are available from the PluggedIn program at <http://pluggedin.palmone.com>

For the Treo 600, the debugger nub is included on the device.

For the Tapwave Zodiac, the debugger nub is included on the device.

Installing the Debugger Nub

As an example, to install the debugger nub on a Tungsten T, follow this process:

1. Using Palm Desktop's Install Tool, add the debugger nub file `DebugNub5-PalmTT.prc` to be installed using a HotSync[®] operation.
2. Place the Tungsten T in a cradle, and press the HotSync button install the PRC file on the device.
3. Once the HotSync operation has completed, press the reset button on the Tungsten T device to restart Palm OS and complete the installation.

Connecting Palm OS Debugger with a Palm OS Garnet Device

To connect Palm OS Debugger with a Palm OS Garnet device, complete the following setup steps:

1. Set Palm OS Debugger's **ARM Palm OS 5 Debug Nub** protocol plug-in preferences.
2. Test the connection between Palm OS Debugger and the device.

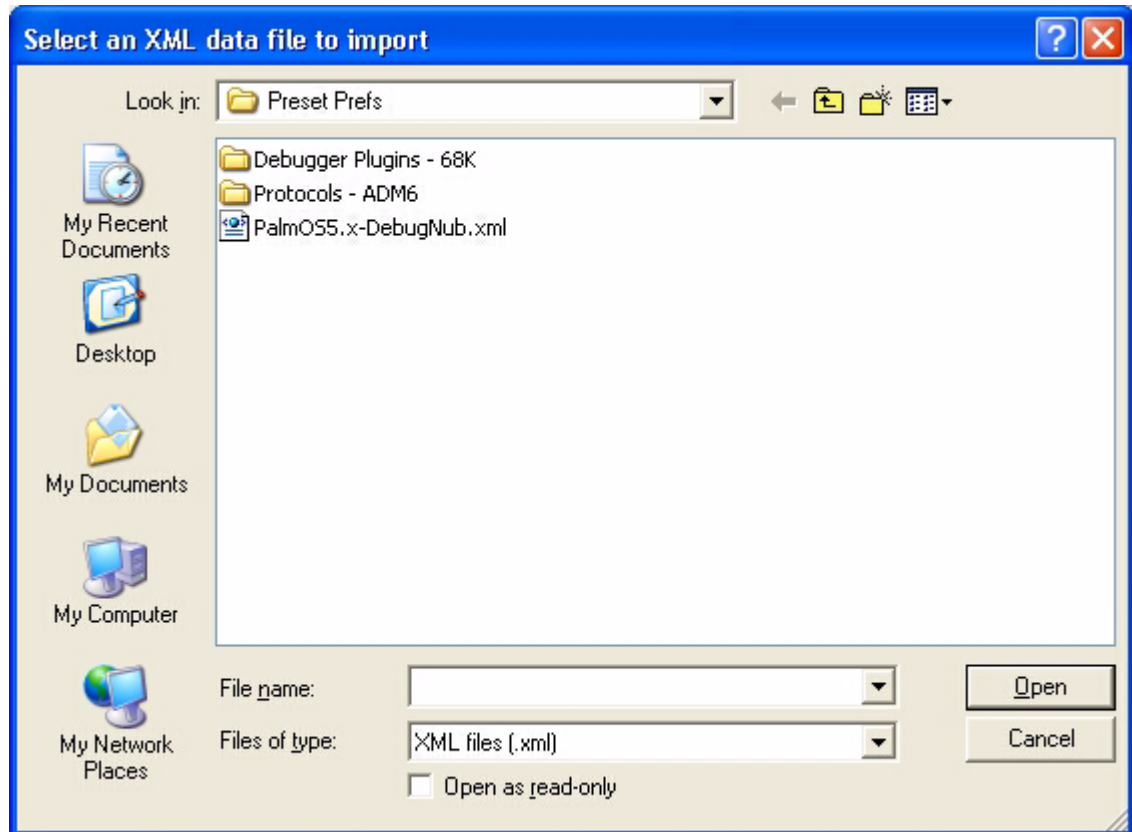
The following sections cover these steps in detail.

1. Setting the ARM Palm OS 5 Debug Nub Plug-In Preferences

To set the **ARM Palm OS 5 Debug Nub** plug-in preferences:

- Open Palm OS Debugger
- Select [Edit](#) > [Preferences](#)
- Click **Import** to import the preset preferences. The dialog box shown in [Figure 2.2](#) on page 9 opens.

Figure 2.2 Import Preferences Dialog Box



- Open the folder `Preset Prefs`, and select the file `PalmOS5.x-DebugNub.xml` in the dialog box. Then click **Open** to set the debugger preferences. If Palm OS Debugger displays a message dialog box asking whether to save modified preferences, click **Yes**.
- To verify the serial communications port settings, select [Edit](#) > [Preferences](#), expand the **Communications** category, and click **Serial**.

2. Testing the Connection

- In Palm OS Debugger, open the Palm OS application (PRC file) to debug by selecting [File](#) > [Open](#).
- Place your Palm OS device in the cradle. Make sure that the cradle is connected properly to the computer.
- Turn on the Palm OS device.

Connecting Palm OS Debugger with a Target

Debugging with Palm OS Garnet Devices

- Enter the debugging shortcut, as shown below, in the input area to enable the debugger nub.



- Select [Control](#) > [Run](#) to start the debugging session.

NOTE: To draw the debugging shortcut, follow these steps:

1. Draw the lowercase, cursive “l” character.
 2. Tap the stylus twice, to generate a dot (a period).
 3. Draw the number 2.
-

For a more detailed description of application debugging, see [“Overview of Application Debugging”](#) on page 17.

Removing the Debugger Nub from a Tungsten T Device

When you are finished using your Tungsten T to debug, you can delete the debug nub by following these steps:

- Perform a “no-notify” reset by holding down the “Up” button on the device while simultaneously pressing the reset button.
- From the Launcher application, select the **Delete** menu item.
- From the Delete dialog, select DebugNub5 - PalmTT and tap **Delete**.

Debugging with Palm OS Garnet Simulator

When debugging with Palm OS Garnet Simulator, the Palm OS Debugger's 68K Palm OS Debug Kernel protocol plug-in communicates through a socket connection with the 68K debugger nub on Palm OS Garnet Simulator.

NOTE: Palm OS Garnet Simulator includes PACE with the 68K debugger nub. You do not need to install a debug nub as part of the setup tasks for debugging with Palm OS Garnet Simulator.

Palm OS Garnet Simulator can be used to debug 68K-based Palm OS applications. You cannot use Palm OS Garnet Simulator to debug native ARM subroutines.

Connecting Palm OS Debugger with Palm OS Garnet Simulator

To connect Palm OS Debugger with Palm OS Garnet Simulator, complete the following setup steps:

1. Set Palm OS Debugger's **68K Palm OS Debug Kernel** protocol plug-in preferences.
2. Set Palm OS Garnet Simulator's communication settings.
3. Test the connection between Palm OS Debugger and Palm Garnet OS Simulator.

The following sections cover these steps in detail.

1. Setting the 68K Palm OS Debug Kernel Plug-In Preferences

To set the **68K Palm OS Debug Kernel** plug-in preferences:

- Open Palm OS Debugger.
- Select [Edit > Preferences](#).
- Click **Import** to import the preset preferences. The dialog box shown in [Figure 2.2](#) on page 9 opens.

Connecting Palm OS Debugger with a Target

Debugging with Palm OS Garnet Simulator

- Open the folder `Preset Prefs`, and select the `PalmOS.68KPoser.xml` preset definition in the dialog box.

Then click **Open** to set the debugger preferences. If Palm OS Debugger displays a message dialog box asking whether to save modified preferences, click **Yes**.

2. Setting Palm OS Garnet Simulator's Communication Settings

In Palm OS Garnet Simulator: Select **Settings > Communication > Communication ports** to bind the 68K debugger transport to `localhost:2000`.

3. Testing the Connection

- In Palm OS Debugger, open the Palm OS application (PRC file) to debug by selecting **File > Open**.
- Enter the debugging shortcut, as shown below, in the input area to enable the debugger nub.



- Select **Control > Run** to start the debugging session.

NOTE: To draw the debugging shortcut, follow these steps:

1. Draw the lowercase, cursive “l” character.
 2. Tap the stylus twice, to generate a dot (a period).
 3. Draw the number 2.
-

For a more detailed description of application debugging, see “[Overview of Application Debugging](#)” on page 17.

Debugging with 68K-Based Devices

When debugging with 68K-based devices, the Palm OS Debugger's 68K Palm OS Debug Kernel protocol plug-in communicates through a serial or USB connection with the 68K debugger nub on the device.

NOTE: Some 68K-based devices do not support debugging over a USB connection. Contact your device manufacturer for more information about your device.

Current 68K-based devices are manufactured with the 68K debugger nub already included. You do not need to install a debug nub as part of the setup tasks for debugging with 68K-based devices.

NOTE: Remember that 68K-based devices can be used to debug only 68K-based Palm OS applications. You cannot use a 68K-based device to debug ARM code.

Connecting Palm OS Debugger with a 68K-Based Device

To connect Palm OS Debugger with a 68K-based device, complete the following setup steps:

1. Set Palm OS Debugger's **68K Palm OS Debug Kernel** protocol plug-in preferences.
2. Test the connection between Palm OS Debugger and the 68K-based device.

The following sections cover these steps in detail.

1. Setting the 68K Palm OS Debug Kernel Plug-In Preferences

To set the 68K Palm OS Debug Kernel plug-in preferences:

- Open Palm OS Debugger
- Select [Edit](#) > [Preferences](#)
- Click **Import** to import the preset preferences. The dialog box shown in [Figure 2.2](#) on page 9 opens.

Connecting Palm OS Debugger with a Target

Debugging with 68K-Based Devices

- Open the folder `Preset Prefs\Debugger Plugins - 68K`, and select the Palm OS 68K preset file dependent on your serial connection:
 - For serial port COM1, select `PalmOS.68KDevice.Serial.COM1.xml` in the dialog box.
 - For serial port COM2, select `PalmOS.68KDevice.Serial.COM2.xml` in the dialog box.
 - For serial port COM3, select `PalmOS.68KDevice.Serial.COM3.xml` in the dialog box.
 - For serial port COM4, select `PalmOS.68KDevice.Serial.COM4.xml` in the dialog box.
 - For USB, select `PalmOS.68KDevice.USB.xml` in the dialog box.

Then click **Open** to set the debugger preferences. If Palm OS Debugger displays a message dialog box asking whether to save modified preferences, click **Yes**.

2. Testing the Connection

NOTE: Before debugging an application over USB, it is best to first perform a HotSync operation to test the USB communication with the device. After you have completed a HotSync operation, you should be able to follow the process described here.

- In Palm OS Debugger, open the Palm OS application (PRC file) to debug by selecting **File > Open**.
- Enter the debugging shortcut, as shown below, in the input area to enable the debugger nub.



NOTE: To draw the debugging shortcut, follow these steps:

1. Draw the lowercase, cursive “L” character.
2. Tap the stylus twice, to generate a dot (a period).
3. Draw the number 2.

-
- Select [Control](#) > [Run](#) to start the debugging session.

If you are using HotSync Manager 4.1 or earlier to debug a 68K application on a 68K device, note that whenever you have to soft reset the device, you will have to re-enter the debugging shortcut. Also, after a soft reset, the USB port does not work immediately and the first HotSync operation has to be cancelled and followed by a second HotSync operation. The second HotSync operation usually succeeds. Only after you are able to complete a HotSync operation will you be able to debug your application.

For a more detailed description of application debugging, see “[Overview of Application Debugging](#)” on page 17.

Debugging with Palm OS Emulator

When debugging with Palm OS Emulator, the Palm OS Debugger’s 68K Palm OS Debug Kernel protocol plug-in communicates through a socket connection with the 68K debugger nub on Palm OS Emulator.

Palm OS Emulator includes the 68K debugger nub. You do not need to install a debug nub as part of the setup tasks for debugging with Palm OS Emulator.

NOTE: Palm OS Emulator can be used to debug only 68K-based Palm OS applications. You cannot use Palm OS Emulator to debug ARM code.

Connecting Palm OS Debugger with Palm OS Emulator

To connect Palm OS Debugger with Palm OS Emulator, complete the following setup steps:

1. Set Palm OS Debugger's **68K Palm OS Debug Kernel** protocol plug-in preferences.
2. Test the connection between Palm OS Debugger and Palm OS Emulator.

The following sections cover these steps in detail.

1. Setting the 68K Palm OS Debug Kernel Plug-In Preferences

To set the **68K Palm OS Debug Kernel** plug-in preferences:

- Open Palm OS Debugger
- Select [Edit](#) > [Preferences](#)
- Click **Import** to import the preset preferences. The dialog box shown in [Figure 2.2](#) on page 9 opens.
- Open the folder `Preset Prefs\Debugger Plugins - 68K`, and select the `PalmOS.68KPoser.xml` preset definition in the dialog box.

Then click **Open** to set the debugger preferences. If Palm OS Debugger displays a message dialog box asking whether to save modified preferences, click **Yes**.

2. Testing the Connection

- In Palm OS Debugger, open the Palm OS application (PRC file) to debug by selecting [File](#) > [Open](#).
- Start Palm OS Emulator, and make sure Emulator is idle, showing the Launcher application.
- Select [Control](#) > [Run](#) to start the debugging session.

For a more detailed description of application debugging, see "[Overview of Application Debugging](#)" on page 17.

Overview of Application Debugging

To start debugging an application in Palm OS Debugger, open the Palm OS application (PRC file) to debug by selecting [File](#) > [Open](#).

68K-Based Application Debugging

To debug a Palm OS 68K application, you need to first connect to a target that can run a 68K application:

- “[Debugging with Palm OS Garnet Devices](#)” on page 7
- “[Debugging with Palm OS Garnet Simulator](#)” on page 11
- “[Debugging with 68K-Based Devices](#)” on page 13
- “[Debugging with Palm OS Emulator](#)” on page 15

Palm OS Debugger displays the application debugging session window, with the corresponding 68K symbolic file information for the Palm OS application you selected when you connected to the debugging target.

The symbolic information is automatically loaded for you if the symbolic file for the Palm OS application is in the same directory as the application’s PRC file. (The symbolic file has the same filename as the PRC file but with the file extension PSYM.) If the symbolic file is moved or renamed, you can load it manually by selecting [Target](#) > [Append Symbolics](#).

If the source files are available, you can set breakpoints in the source view. (Palm OS Debugger may not be connected to the debug target yet, so the breakpoints may be marked as unresolved.)

To connect Palm OS Debugger with the debug target, select [Control](#) > [Run](#) to start the debugging session:

- If the 68K debug console is already active, Palm OS Debugger connects with the debug target.
- If the 68K debug console is not active, Palm OS Debugger prompts you to activate the debug console by entering shortcut-2:



Connecting Palm OS Debugger with a Target

Overview of Application Debugging

NOTE: To draw the debugging shortcut, follow these steps:

1. Draw the lowercase, cursive “L” character.
 2. Tap the stylus twice, to generate a dot (a period).
 3. Draw the number 2.
-

Once Palm OS Debugger has connected with the debug console, Palm OS Debugger loads the application on the debug target. The debug target launches the application.

NOTE: Due to a problem in the 68K Serial Link Manager for the Palm m500 device, the stop and kill commands may not initially respond when you debug a 68K application using an m500 device. To correct this problem, turn off the m500 device and then turn it on again.

ARM Subroutine Debugging

ARM subroutines, also called PACE native objects (PNO), are typically built and written to files of the following form:

`ARM*####.bin`

where “*” is any character, and “####” is a hexadecimal representation of the ARM subroutine resource ID, or

`ARM*####.sbin`

for subroutines that are larger than 64 KB.

If Palm OS Debugger finds code resources of type “ARM*” in the Palm OS application, then Palm OS Debugger looks for corresponding symbolic file of the form `ARM*####.bin.elf` (or `ARM*####.sbin.elf`) in the same directory as the Palm OS application’s PRC file. If Palm OS Debugger finds an ARM symbolic file, then Palm OS Debugger opens an ARM debug session window.

Interaction Between 68K and ARM Debug Windows

Palm OS Debugger presents debug information for 68K and ARM code in separate debug windows. To a certain extent, Palm OS

Debugger treats the two components as if they are being run on two different CPUs, though they are not completely independent.

When a break or crash occurs, the debug session window corresponding to the code that received the break or crash becomes the front-most window. The status in the lower-right corner of the window indicates that the code execution is stopped.

The 68K debug window is a representation of PACE running on the debug target. PACE, the Palm Application Compatibility Environment, not only emulates a 68K processor executing the 68K code, but also provides support for all 68K debug services.

PACE is an ARM application. Any time the ARM session is stopped, the 68K session is implicitly stopped as well, even if the debug window status indicates that the 68K session is running. When the ARM session is resumed, PACE and the 68K session continue to execute starting from where the session was implicitly stopped.

When debugging with both 68K and ARM sessions, you should be able to stop either session by clicking the **Stop** icon in the toolbar or using the **Stop** menu item. For interactive debugging, it is best to stop and resume execution in only one session window at a time:

If your code causes a crash in the ARM debug nub but not the 68K debug console, you may be able to use the 68K session window to determine what the 68K code was doing prior to the crash. To investigate, select the 68K session window, stop execution, and examine the stack trace, variables, registers, and memory values displayed.

If you can recover from the crash in the ARM session by changing the program counter or a register, then you may be able to resume execution as well.

Palm OS Protein Application Debugging

To debug a Palm OS Protein application, you need to first connect to a target that can run a Palm OS Protein application. That is, you need to connect to a device that is running Palm OS Cobalt or later.

At the time this book was written, there were no targets that support running Palm OS Protein applications.

Connecting Palm OS Debugger with a Target

Overview of Application Debugging

Setting Palm OS Debugger Preferences

This chapter discusses Palm OS Debugger preferences.

The easiest way to set preferences is to import preset values from a preferences XML file. However, this chapter describes the preferences in detail in case you need to set specific preferences for debugging scenarios not covered by the preset value files.

This chapter covers the following topics:

- [“How to Set Preferences”](#) on page 22
- [“Importing and Exporting Preferences”](#) on page 24
- [“Communications Preferences”](#) on page 25
- [“Debugger Plug-in Preferences”](#) on page 27
- [“Debugger UI Preferences”](#) on page 32
- [“Disassembler Preferences”](#) on page 34
- [“Download Plug-ins Preferences”](#) on page 36
- [“Protocols Preferences”](#) on page 37
- [“Runtime Helpers Preferences”](#) on page 39
- [“Symbolics Preferences”](#) on page 41

Setting Palm OS Debugger Preferences

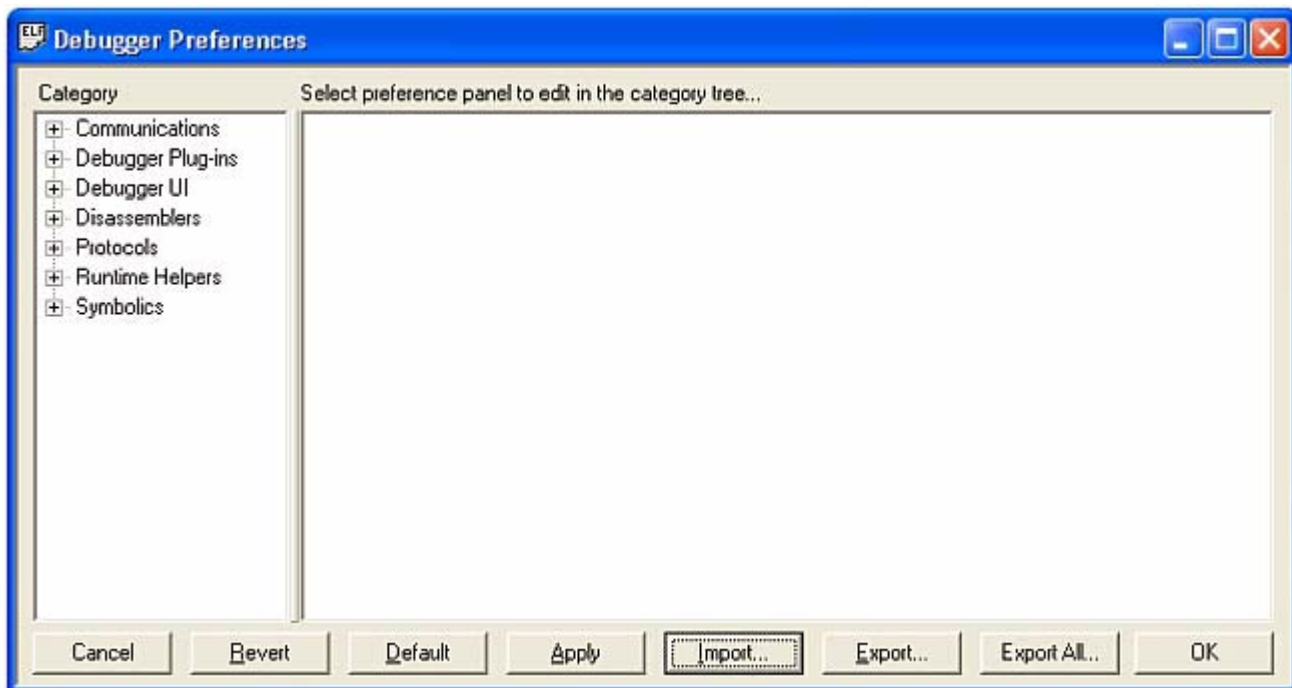
How to Set Preferences

How to Set Preferences

To set Palm OS Debugger preferences:

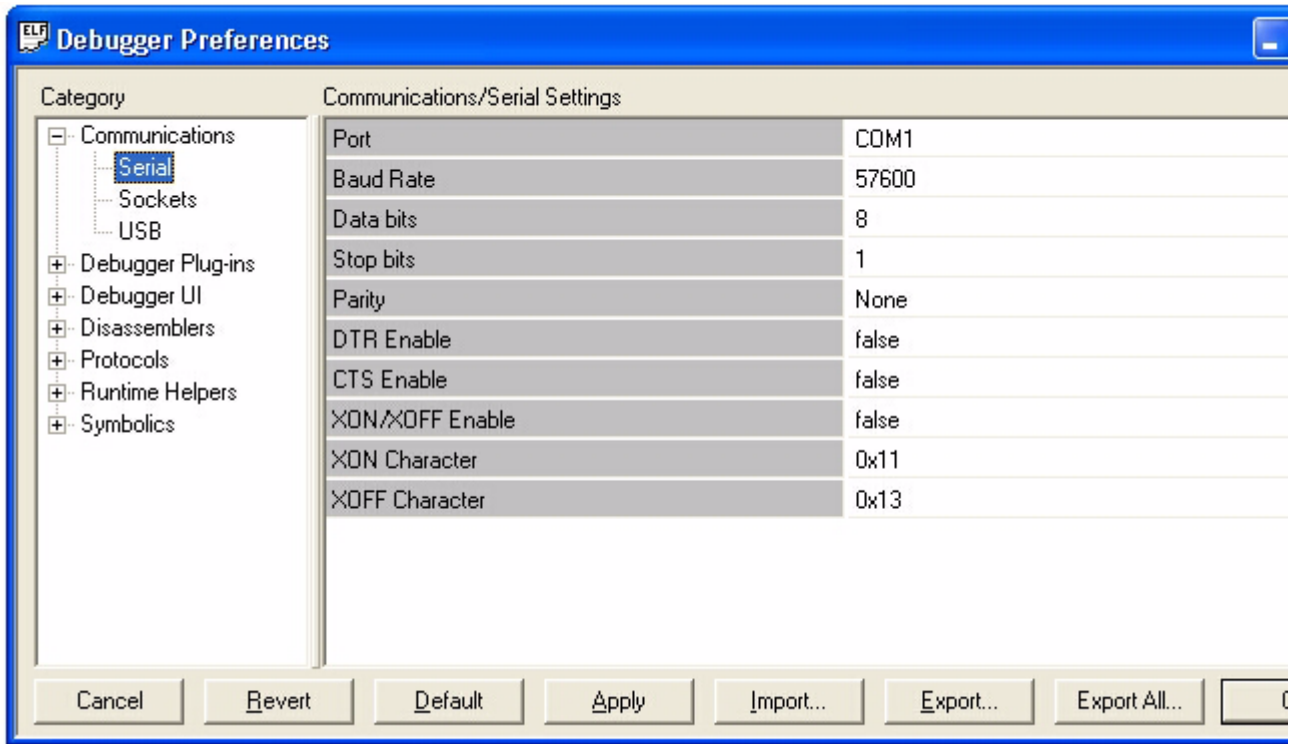
1. From the **Edit** menu, select **Preferences**. This opens the Debugger Preferences dialog box.

Figure 3.1 Debugger Preferences Dialog Box



2. Select a category from the category tree, click the plus sign (+) next to it to expand it, and select a subcategory. This opens a preference panel for that subcategory.

Figure 3.2 Debugger Preferences dialog box with the Serial subcategory selected



3. Edit the settings.
4. Click **Apply** to save your changes, or click **Revert** to discard the changes. (You also have the option of clicking **Cancel** to discard the changes, but clicking **Cancel** also closes the Debugger Preferences window.)
5. Select other categories and edit their preference panels.
6. To return a panel to its default settings, open the panel and click **Default**.
7. When you are finished setting preferences, click **OK** to close the Debugger Preferences dialog box.

Setting Palm OS Debugger Preferences

Importing and Exporting Preferences

NOTE: In the panel where the preference settings are displayed, you can resize the caption field and the value field. For example, if the field is too short and the contents of the field do not display completely, you can make the field longer. To do this, use the resize bar located between the caption field and the value field. Drag the resize bar to the left or right to expose the contents of either field.

Importing and Exporting Preferences

Palm OS Debugger offers pre-set preferences files, written in XML format, that you can use to set up the appropriate preferences for debugging. You can also create your own pre-set preferences files.

Importing

To use pre-set preferences, use the **Import** button in the Debugger Preferences window.

To import pre-set preferences:

1. In the Debugger Preferences window, click the **Import** button. This opens a dialog box entitled *Select an XML data file to import*.
2. Browse to the directory `Preset Prefs` and select the file whose name corresponds to the debug target you are using.

Exporting

You can create your own pre-set preferences files.

Exporting Preferences from One Preferences Panel

To create a pre-set preferences file containing the preferences from one preferences panel:

1. In the Debugger Preferences window, select an item from the category tree to display the preferences panel containing the preferences you want to export.
2. Click the **Export** button to open a *Save as* dialog box.

3. Browse to the location for your preferences file.
4. Type a name for the preferences file.
5. Click **Save** to save your preferences in an XML file.

Exporting All Palm OS Debugger Preferences

To create a pre-set preferences file containing *all* your Palm OS Debugger preferences:

1. In the Debugger Preferences window, click the **Export All** button to open a *Save as* dialog box.
2. Browse to the location for your preferences.
3. Type a name for the preferences file.
4. Click **Save** to save all your Palm OS Debugger preferences in an XML file.

Communications Preferences

To set Communications Plug-in preferences, select **Communications** from the category tree, and then select **Serial** or **Sockets**.

The default settings in this panel are the correct settings for debugging with a Palm OS Garnet or Palm OS Cobalt device over a serial connection.

Serial Preferences

If you are connecting a device to Palm OS Debugger through a serial connection, set these preferences.

Port

Default setting is *COM1*.

Baud Rate

Default setting is *57600*.

Data bits

Default setting is *8*.

Setting Palm OS Debugger Preferences

Communications Preferences

Stop bits

Default setting is *1*.

Parity

Default setting is *None*.

DTR Enable

Default setting is *false*, which means that Data Terminal Ready (DTR) is not enabled.

CTS Enable

Default setting is *false*, which means that Clear To Send (CTS) is not enabled.

XON/XOFF Enable

Default setting is *false*, which means that XON/XOFF (special characters that you can insert into the byte stream) is not enabled.

XON Character

Default setting is *0x11*.

XOFF Character

Default setting is *0x13*.

Sockets Preferences

If you are connecting to Palm OS Debugger through a sockets connection, set these preferences. This plug-in communicates over standard sockets to the IP address and port that you specify here.

NOTE: These settings are available for connecting to Palm OS Simulator or Palm OS Emulator. Note that no Palm OS Emulator ROM is available for Palm OS Garnet or Palm OS Cobalt.

IP Address

Default setting is *127.0.0.1*, which is the setting to use to connect to a Palm OS Emulator or Palm OS Simulator session running on the same computer that is running Palm OS Debugger.

Port

Default setting is *2000*.

Debugger Plug-in Preferences

To set Debugger Plug-in preferences, select **Debugger Plug-ins** from the category tree, and then select **68K** or **ARM**.

The Debugger Plug-in is a parent of the Protocols Plug-in (for information on Protocol settings, see “[Protocols Preferences](#)” on page 37).

68K Preferences

Set these preferences for general 68K debugging settings that affect all 68K-related protocols. Currently, the only protocol available for 68K debugging is the *68K Palm OS Debug Kernel Protocol Plug-in*.

This plug-in is used to communicate with 68K-based devices or with ARM-based devices through PACE. (PACE is the Palm OS Application Compatibility Environment. For more information about PACE, see *Palm OS Simulator Guide*.)

Protocol

Default setting is *68K Palm OS Debug Kernel Protocol Plug-in*. This is the only valid setting for 68K debugging.

Max stack frames

Default setting is *100*.

When you are doing a stack trace, if the stack crawl is taking a long time, you can limit the maximum number of stack frames that get displayed to improve the stack crawl speed.

Setting Palm OS Debugger Preferences

Debugger Plug-in Preferences

Cache stack backtrace

Default setting is *false*, which means the Stack Trace window (see “[Stack Trace View](#)” on page 70) does not reuse information from the previous stack trace.

If you set this to *true*, the Stack Trace window may reuse information from the previous stack trace, if you are still in the same function the next time you stop someplace in the code.

Stop at entry point

Default setting is *true*, which means Palm OS Debugger stops at the entry point of your program, as if you had set a breakpoint there. (You can specify your application entry point under *Protocol Preferences > 68K Palm OS Debug Kernel Preferences*. See “[Application Entry Point](#)” on page 37.)

ARM Preferences

Set these preferences if you are connecting to an ARM device.

NOTE: Rather than setting these preferences manually, you can import preset preferences for most protocols. For details, see “[Importing and Exporting Preferences](#)” on page 24.

Protocol

Select the protocol that corresponds to the type of connector you are using. The only valid option is *ARM Palm OS 5 Debug Nub*.

Real Time OS Plug-in

Default setting is *None*. Palm OS Debugger uses the other values for JTAG debugging. These values are not valid for application debugging with debugging nubs such as the 68K-based Debug Manager, PACE, or the ARM-based Debug Manager.

Palm OS Cobalt users also have the option of selecting *Palm OS 6.0*, which means Palm OS Debugger keeps track of all the processes and threads currently running in Palm OS Cobalt.

Substitute SB for R9

Default setting is *true*. Do not change this setting if you are doing Palm OS development.

The *true* setting means that Static Base (SB) is substituted for register 9 (R9). Normally, when the symbolics file contains global variables, the global variables are reported as being in memory at R9 plus an offset. Because the ARM-based Palm OS shared library model relocates global variables using a complex expression, set this preference to *true* to substitute the Static Base for R9.

SB expression

Default setting for Palm OS Garnet is *genreg_pc sect_base constu(8) plus deref dup constu(0xFFFF) and swap constu(16) ror constu(0xFFFF) and constu(5) rol plus genreg_gp1 deref plus deref*.

Default setting for Palm OS Cobalt is *genreg_pc sect_base constu(8) plus deref*.

Do not change this setting if you are doing Palm OS development. This setting specifies the expression that Palm OS Debugger uses to compute the Static Base.

Max stack frames

Default setting is *100*.

When you are doing a stack trace, if the stack crawl is taking a long time, you can limit the maximum number of stack frames that get displayed to improve the stack crawl speed.

Cache stack backtrace

Default setting is *false*, which means the Stack Trace window (see [“Stack Trace View”](#) on page 70) does not reuse information from the previous stack trace.

If you set this to *true*, the Stack Trace window may reuse information from the previous stack trace, if you are still in the same function the next time you stop someplace in the code.

Correct stack backtrace return address

Default setting is *true*.

Setting Palm OS Debugger Preferences

Debugger Plug-in Preferences

When you are doing a stack crawl, the return address often points to the instruction following the instruction that called the current subroutine. Setting this preference to *true* will attempt to correct stack backtrace return addresses to show the actual instruction which called the stack function.

Setting this preference to *false* will disable this feature.

Launch action

This setting specifies what Palm OS Debugger does when you click **Run** for the first time.

The options are:

- *Set PC to entry point and halt*
- *Set PC to entry point and run*
- *Set PC to reset vector and halt* (the reset vector = 0, or 0xffff0000)
- *Set PC to entry point and run*
- *Soft reset*
- *Hard reset*
- *Nothing*

For debugging applications with the Palm OS Garnet ARM debug nub, set Launch action to *Nothing*.

Skip downloading code to target

Default setting is *false*, though an imported setting may change this value to *true*.

If the code has already been downloaded to the target, you should set this to *true* so that Palm OS Debugger does not download ARM-based code to the ARM debug nub. (Downloading 68K-based code is handled by the 68K plug-in.)

Allow ROM address overriding

Default setting is *false*.

This setting is not used for application debugging.

Download Helper Plug-in

Default setting is *None*. This setting is not used for application debugging.

Default CPU when unknown

Default setting is *ARM*. The other option is *Thumb*.

Symbolics files generally tell Palm OS Debugger exactly which parts of the code are ARM and which parts of the code are Thumb. Palm OS Debugger needs to know which type of code is at a specific address to set breakpoints correctly (breakpoint instructions differ for ARM and Thumb).

This preference specifies how Palm OS Debugger treats code when there are no symbolics for a specific address.

Type/Creator section name

Default setting is *.text*. In most cases, you should not change this setting.

Palm OS Debugger uses this setting with the **Type/Creator section offset** setting.

Type/Creator section offset

Default setting is *20*. Do not change this setting.

The **Type/Creator section offset** and **Type/Creator section offset** settings tell Palm OS Debugger where to find a shared library's type, creator ID, resource, and resource ID information in an ELF file. Palm OS Debugger uses these settings to process shared library load and unload notifications, and to enable and disable breakpoints in a shared library.

Each Palm OS shared library has unique shared library data in the initial bytes of the code section. By default, the code section in the ELF file is named *.text*.

Auto symbolic load directory 1

Default setting is to leave this field blank.

Set this field to indicate which directory Palm OS Debugger should recursively scan first in order to find symbolic files.

Setting Palm OS Debugger Preferences

Debugger UI Preferences

Auto symbolic load directory 2

Default setting is to leave this field blank.

Set this field to indicate a subsequent directory Palm OS Debugger should recursively scan in order to find symbolic files.

Auto symbolic load directory 3

Default setting is to leave this field blank.

Set this field to indicate a subsequent directory Palm OS Debugger should recursively scan in order to find symbolic files.

Memory mapped register definition file

Default setting is to leave this field blank. If you have created a Memory Mapped Definition file, specify the name and path here.

A Memory Mapped Definition file is an XML file describing all memory mapped registers that are available for a target. These can also be used to describe statically located global variables that are not described in symbolics. For more information and complete XML format documentation, see the Readme file at:

SDK/tools/Palm OS Debugger/Memory Mapped
Registers/ReadMe.txt

Also see the sample XML file at:

SDK/tools/Palm OS Debugger/Memory Mapped
Registers/SampleDefs.xml

Debugger UI Preferences

To set UI preferences, select **Debugger UI** from the category tree, and then select **Fonts** or **Session**.

Fonts Preferences

Select a font for the data displayed in the Palm OS Debugger windows.

Source view

Default setting is *Courier New, 10pt*.

Files view

Default setting is *MS Sans Serif, 9pt.*

Breakpoints view

Default setting is *MS Sans Serif, 9pt.*

Registers view

Default setting is *Courier New, 9pt.*

Variables view

Default setting is *Courier New, 9pt.*

Global Variables view

Default setting is *Courier New, 9pt.*

Memory view

Default setting is *Courier New, 9pt.*

Stack Trace view

Default setting is *Courier New, 9pt.*

Consoles

This sets the font used to display output in the **STDIO Console** window and the **Debug Console** window.

Default setting is *Courier New, 9pt.*

Session Preferences

Automatically load last executable on application launch

Default setting is *false.*

Automatically run last executable on application launch

Default setting is *false.*

Automatically restore session settings for executables

Default setting is *true.*

Setting Palm OS Debugger Preferences

Disassembler Preferences

Palm OS Debugger stores sessions settings (a list of symbolic files and breakpoints) for each executable. The session settings file has the same filename as the executable file with the file extension PUD.

Using the value *true* for this setting means that the PUD file for your executable is loaded automatically when you run the executable.

Automatically restore session window positions

Default setting is *true*.

Tab size

Default setting is 4.

Disassembler Preferences

To set Disassembler Plug-in preferences, select **Disassemblers** from the category tree, and then select **68K** or **ARM**.

68K Preferences

If you are debugging 68K code, set these preferences.

Disassembler engine

Default setting is Palm OS Debugger. The other option is *Palm Debugger*.

The Palm OS Debugger disassembler is the recommended disassembler for Palm OS Debugger users. If you have problems with the Palm OS Debugger disassembler, however, you can choose to use the *Palm Debugger* disassembler, which is the disassembler from the Palm Debugger tool. (For an explanation of the difference between Palm OS Debugger and the Palm Debugger, see "[How Does Palm OS Debugger Compare to Palm Debugger?](#)" on page 2.

Uppercase opcodes hex bytes

Default setting is *false*, which means that alphabetic characters in hexadecimal opcode bytes are displayed as lowercase letters.

Use the option *true* to display alphabetic characters in hexadecimal opcode bytes as uppercase letters.

Uppercase mnemonics operands

Default setting is *false*, which means that operands in hexadecimal numbers are displayed as lowercase letters.

Use the option *true* to display operands in hexadecimal numbers as uppercase letters.

Show offset values in hex

Default setting is *false*, which means Palm OS Debugger displays offsets as decimal numbers.

The hexadecimal value of offsets are often directly visible in the opcode bytes, so signed decimal numbers is the default display type for offsets.

Set this preference to *true* to display offsets as hexadecimal numbers.

Show immediates values as hex

Default setting is *false*, which means Palm OS Debugger displays arithmetic immediates as decimal numbers.

The hexadecimal value of intermediates are often directly visible in the opcode bytes, so decimal numbers is the default display type for arithmetic intermediates.

Set this preference to *true* to display arithmetic immediates as hexadecimal numbers.

Lookup names for addresses

Default setting is *true*, which means that if you are branching to another function, Palm OS Debugger displays the function name rather than the address.

Set this preference to *false* for slightly faster performance.

ARM Preferences

If you are debugging ARM code, set these preferences.

Setting Palm OS Debugger Preferences

Download Plug-ins Preferences

Uppercase opcodes

Default setting is *false*, which means that alphabetic characters in hexadecimal opcode bytes are displayed as lowercase letters.

Use the option *true* to display alphabetic characters in hexadecimal opcode bytes as uppercase letters.

Uppercase operands

Default setting is *false*, which means that operands in hexadecimal numbers are displayed as lowercase letters.

Use the option *true* to display operands in hexadecimal numbers as uppercase letters.

Show offsets in hex

Default setting is *false*, which means Palm OS Debugger displays offsets as decimal numbers.

The hexadecimal value of offsets are often directly visible in the opcode bytes, so signed decimal numbers is the default display type for offsets.

Set this preference to *true* to display offsets as hexadecimal numbers.

ARM architecture version

Default setting is *v4T*.

Set this to the ARM architecture version that is appropriate for your ARM chip. The disassembler needs this information in order to disassemble the opcodes correctly.

Download Plug-ins Preferences

These preferences are used for debugging with prototype development boards, and are not used for Palm OS application debugging.

Protocols Preferences

To set Protocols Plug-in preferences, select **Protocols** from the category tree, and then select **68K Palm OS Debug Kernel** or **ARM Palm OS 5 Debug Nub**.

68K Palm OS Debug Kernel Preferences

This is the kernel that talks to Palm OS Emulator or to a 68K device.

Debug target select

Default setting is *Palm Device*. You can select *Poser* to indicate Palm OS Emulator, but note that Palm OS Emulator cannot be used with ARM-based code.

Communications Plug-in

Default setting is *Win32 Serial Communications Plug-in*. This is currently the only supported value if you are connecting to a 68K device.

If you are connecting to Palm OS Emulator, you should set this option to *Win32 Sockets Communications Plug-in*. Make sure the Sockets communication plug-in preferences are set correctly to 127.0.0.1 on port 2000.

Path to poser

Default setting is to leave this field blank; this preference is optional.

Palm OS Debugger uses this setting to automatically launch Palm OS Emulator, if Emulator is not already running.

Command line arguments to poser

Default setting is to leave this field blank.

If you are connecting to Palm OS Emulator, you can specify command line arguments here.

Application Entry Point

Default setting is *PilotMain*. You can set this to any valid function name.

Setting Palm OS Debugger Preferences

RTOS Plug-ins Preferences

ARM Palm OS 5 Debug Nub Preferences

These preferences are for connecting to an ARM device's debug nub.

Communications Plug-in

Default setting is *Win32 Serial Communications Plug-in*.

The other option is *Win32 Sockets Communications Plug-in*.

Log protocol details

Default setting is *Log errors to Debug Console*. This setting is not necessary, so you can disable it.

RTOS Plug-ins Preferences

These preferences are used for debugging with prototype development boards, and are not used for Palm OS application debugging.

To set real time operating system plug-in preferences, select **RTOS Plug-ins** from the category tree, and then select **Palm OS 6.0**.

Palm OS 6.0 Preferences

These preferences are for process- and thread-aware debugging of Palm OS Cobalt applications.

Stop on Thread Created

Default setting is *false*.

Setting this preference to *true* means Palm OS Debugger suspends program execution when a new thread is created.

Stop on Thread Faulted

Default setting is *true*. This setting means that Palm OS Debugger suspends program execution when a thread is faulted.

Stop on Thread Destroyed

Default setting is *false*.

Setting this preference to *true* means Palm OS Debugger suspends program execution when a thread is destroyed.

Stop on Process Created

Default setting is *false*.

Setting this preference to *true* means Palm OS Debugger suspends program execution when a new process is created.

Stop on Process Destroyed

Default setting is *false*.

Setting this preference to *true* means Palm OS Debugger suspends program execution when a process is destroyed.

Stop on Library Loaded

Default setting is *false*.

Setting this preference to *true* means Palm OS Debugger suspends program execution when a library is loaded.

Stop on Library Unloaded

Default setting is *false*.

Setting this preference to *true* means Palm OS Debugger suspends program execution when a library is unloaded.

Target CPU is XScale

Default setting is *true*.

Runtime Helpers Preferences

These preferences are used for debugging with prototype development boards, and are not used for Palm OS application debugging.

To set Runtime Helpers preferences, select **Runtime Helpers** from the category tree, and then select **ARM Palm OS**.

Setting Palm OS Debugger Preferences

Runtime Helpers Preferences

NOTE: If you change runtime preferences, you may cause an adverse affect on runtime performance.

ARM Palm OS Preferences

These preferences are for ARM targets.

Enable Palm Runtime Helper

Default setting is *false*.

For Palm OS Garnet or Palm OS Cobalt, set this to *true*.

DbgPostLoad func name or address

Default setting is *DbgPostLoad*. Normally, you should not change this.

This lets Palm OS Debugger detect where shared libraries are loaded.

DbgPostUnload func name or address

Default setting is *DbgPostUnload*. Normally, you should not change this.

This lets Palm OS Debugger detect where shared libraries are unloaded.

DbgBreak func name or address

Default setting is *HALDbgBreak*. Normally, you should not change this. This is a function that is in a shared library in Palm OS; you can use it as a breakpoint. It is like a breakpoint that you can compile into your code.

Binary file 1

Default setting is to leave this field blank.

Binary file 1 address

Default setting is to leave this field blank.

Binary file 2

Default setting is to leave this field blank.

Binary file 2 address

Default setting is to leave this field blank.

Symbolics Preferences

Palm OS Debugger uses the DWARF symbolics format. Select **Symbolics** from the category tree, and then select **DWARF 1.1** or **DWARF 2.0**.

DWARF 1.1 Preferences

Normally, you do not need to change any of these preferences. The ones you most like may change are **Relocate Palm libraries** and **Type/Creator section name**.

Byte size of FT_char, FT_signed_char, FT_unsigned_char

Default setting is 1. Do not change.

Byte size of FT_short, FT_signed_short, FT_unsigned_short

Default setting is 2. Do not change.

Byte size of FT_integer, FT_signed_integer, FT_unsigned_integer

Default setting is 4. Do not change.

Byte size of FT_long, FT_signed_long, FT_unsigned_long

Default setting is 4. Do not change.

Byte size of FT_long_long, FT_signed_long_long, FT_unsigned_long_long

Default setting is 8. Do not change.

Byte size of FT_point, FT_label

Default setting is 4. Do not change.

Setting Palm OS Debugger Preferences

Symbolics Preferences

Byte size of FT_float

Default setting is 4. Do not change.

Byte size of FT_dbl_prec_float

Default setting is 8. Do not change.

Byte size of FT_ext_prec_float

Default setting is 10. Do not change.

Byte size of FT_complex

Default setting is 8. Do not change.

Byte size of FT_dbl_prec_complex

Default setting is 16. Do not change.

Byte size of FT_ext_prec_complex

Default setting is 32. Do not change.

Byte size of FT_void

Default setting is 1. Do not change.

Byte size of FT_boolean

Default setting is 1. Do not change.

Relocate PALM libraries

Default setting is *true*. This must be set to *true* in order for Palm OS Debugger to be able to debug shared libraries.

Type/Creator section name

Default setting is *.text*. In most cases, you should not change this setting.

Palm OS Debugger uses this setting with the **Type/Creator section offset** setting.

Type/Creator section offset

Default setting is 20. Do not change this setting.

The **Type/Creator section offset** and **Type/Creator section offset** settings tell Palm OS Debugger where to find a shared library's type, creator ID, resource, and resource ID information in an ELF file. Palm OS Debugger uses these settings to process shared library load and unload notifications, and to enable and disable breakpoints in a shared library.

Each Palm OS shared library has unique shared library data in the initial bytes of the code section. By default, the code section in the ELF file is named `.text`.

DWARF 2.0 Preferences

The DWARF 2.0 plug-in omits any `TAG_member` tags that do not have locations for the members.

Relocate PALM libraries

Default setting is *true*. This must be set to *true* in order for Palm OS Debugger to be able to debug shared libraries.

Type/Creator section name

Default setting is *.text*. In most cases, you should not change this setting.

This setting tells Palm OS Debugger where to find a shared library's type and creator ID. Each Palm OS shared library has a chunk of data at the beginning of a certain section; typically, this section is named `*.text`.

Type/Creator section offset

Default setting is *20*. Do not change this setting.

Omit artificial variables

Default setting is *true*. This makes the Variables view more compact; Palm OS Debugger does not display any artificial variables. For information on the Variables view, see "[Variables View](#)" on page 64.

Omit variables that start with

Default setting is to leave this field blank. To not to display any variables that start with a particular string, specify that string here.

Setting Palm OS Debugger Preferences

Symbolics Preferences

Running Palm OS Debugger

This chapter describes each of the Palm OS Debugger windows and discusses how to use them. It covers the following topics:

- An overview of Palm OS Debugger functionality. See “[Palm OS Debugger Overview](#)” on page 46.
- Instructions for getting started. See “[Getting Started](#)” on page 47.
- A detailed discussion of each of the Palm OS Debugger windows:
 - “[Source View](#)” on page 48
 - “[Files View](#)” on page 55
 - “[Breakpoints View](#)” on page 58
 - “[Registers View](#)” on page 62
 - “[Variables View](#)” on page 64
 - “[Global Variables View](#)” on page 66
 - “[Memory View](#)” on page 67
 - “[Processes View](#)” on page 69
 - “[Stack Trace View](#)” on page 70
 - “[Expressions View](#)” on page 70
 - “[Profiler View](#)” on page 73
 - “[STDIO Console](#)” on page 73
 - “[Debug Console](#)” on page 75

Palm OS Debugger Overview

You can use Palm OS Debugger to:

- View information about applications and their source files.
- Set breakpoints.
- Display a list of registers and their values.
- View information about variables.
- View information about memory address locations and their contents.
- View processes that Palm OS Debugger knows about.
- View the stack trace.
- Evaluate C expressions.
- View standard input and output operations (such as debug messages) in an STDIO console.
- Interact with your debugging session in a command line environment.

You can open an instance any of the Palm OS Debugger tabs in its own separate window. To do this, select **Windows** in the Palm OS Debugger menu bar, and then select a view from the list of views.

Getting Started

NOTE: While you are following the steps described here, whenever Palm OS Debugger displays a dialog asking "Do you want to save modified preferences?" you should click Yes.

Modifying Preferences

To modify Palm OS Debugger preferences, follow the instructions in [Chapter 3](#), "[Setting Palm OS Debugger Preferences](#)," on page 21.

Using the Palm OS Debugger Windows

This section explains how to use each of the Palm OS Debugger windows (also called tabs).

Source View

The source view shows you your application source file in the specified mode: either C/C++, Assembly, or mixed. In the source view, you can read your source code and graphically view any breakpoints in your code.

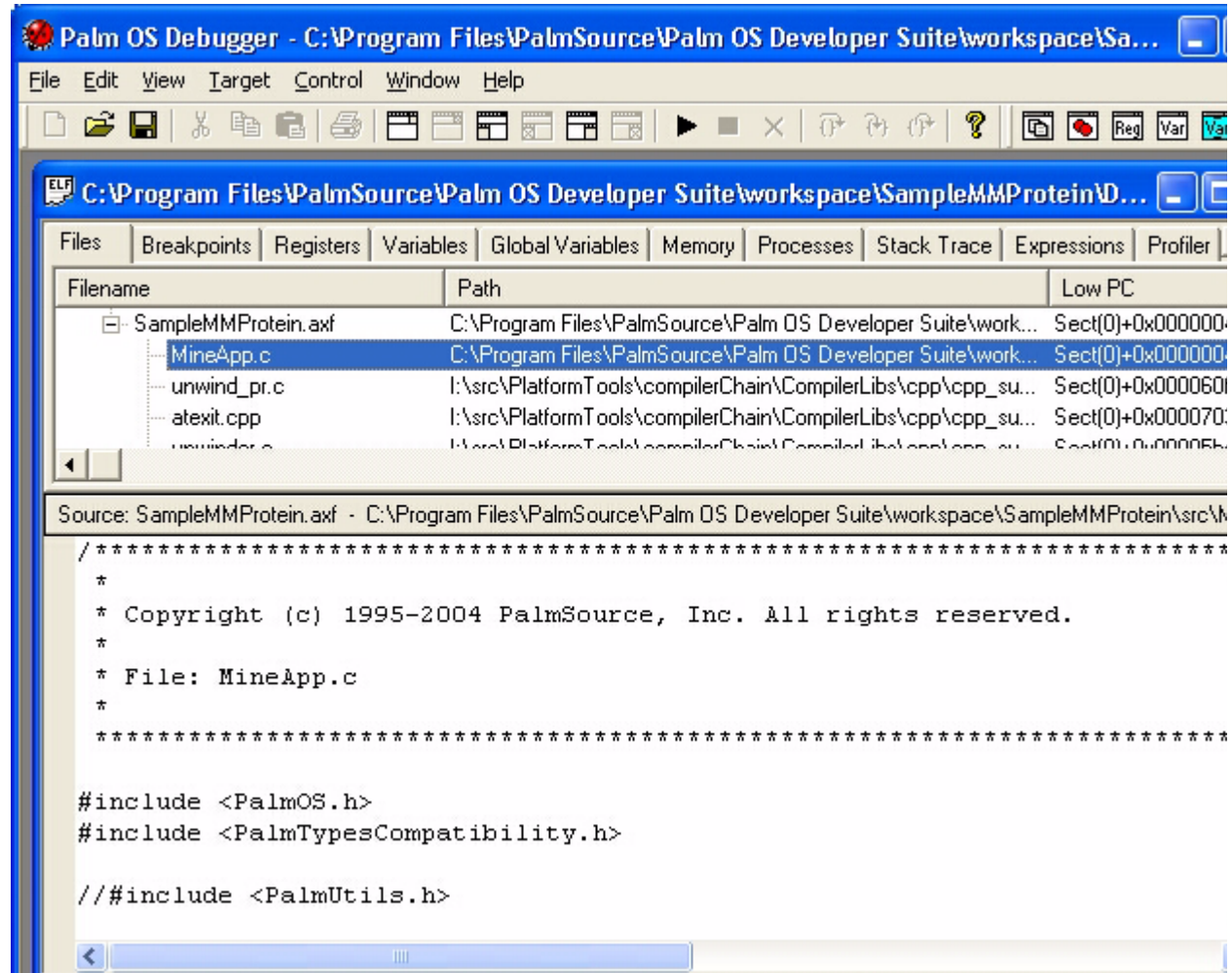
The source view pane is the bottom pane of every view described below, and the user interface of the source view is maintained across every view.

Note that, by default, the source view shows the current location in the source based on the program counter. But the content and current location can be changed in several ways:

- When you select a breakpoint in the breakpoint window, the source view shows the breakpoint you just selected.
- When you select an item from the call stack, the source view shows the item you just selected.
- When you select a file in the files view, the source view shows you the selected file.
- When the program counter is within an area of memory where you have no symbolic information available, the source view shows an assembly view of the code.

If you change the file view, you can get back to the current program counter location by using the **Goto PC** command. (To view the menu, right-click in the source view window.)

Figure 4.1 Source View



Source View User Interface

This pane is available within all views in the debugger. When you open Palm OS Debugger, the pane displays the contents of the file selected in the [Files View](#). When you are debugging, the pane shows the current program counter location in the source or assembly. It is always the lowest pane, but you can resize the pane.

The lower pane in the source view is known as the source view pane. It is a resizable pane that displays the contents of the selected source file; if no source file is selected in the files view, then this pane is empty. In this pane, you can use the following functions:

Running Palm OS Debugger

Using the Palm OS Debugger Windows

- View your source code.
- Use **Find**, **Find Next**, and **Find Previous** to perform searches.
- **Select** and **Copy** text to the clipboard.
- Set and clear breakpoints.
- Move to the next code line where you can place a breakpoint, or back up to the next prior code line where you can place a breakpoint.

Setting Breakpoints in the Source View

The source view uses gray squares in the left margin to indicate where you can set a breakpoint. You can only set breakpoints on code lines that have a gray square in the left margin.

TIP: If your breakpoint boxes are not lining up or if they appear to be on incorrect lines such as comment lines, check that your source file and symbolic file are both up-to-date.

- To set a breakpoint, click the gray square.
 - If the symbolic file containing the new breakpoint has not yet been loaded, then a white dot appears indicating that the breakpoint is set but unresolved. In the breakpoints view, the status for this breakpoint is listed as “Unresolved.”
 - If the symbolic file has already been loaded, then a red dot appears indicating that the breakpoint is both set and enabled. In the breakpoints view, the status for this breakpoint is listed as “Enabled.”
- To run the code up to a breakpoint, SHIFT-click a gray square that shows a red dot.
- To run to a given source line without setting a permanent breakpoint, SHIFT-click that source line’s gray box to set a “one time” breakpoint. This function is handy for getting out of long loops without having to set a breakpoint, run to the breakpoint, and then clear the breakpoint.
- To disable a breakpoint, CTRL-click a gray square that shows a red dot. The red dot changes to a white dot, indicating that

the breakpoint is set but disabled. In the breakpoints view, the status for this breakpoint is listed as “Disabled.”

- To clear a breakpoint, click a gray square that shows a white or a red dot. The dot disappears, indicating that the breakpoint has been cleared.

Note that each line of C source corresponds to a range of possibly discontinuous addresses. When you set a breakpoint in C source mode, the breakpoint is set at the beginning of the range of addresses.

In mixed mode or assembly mode, you can set one or more breakpoints within the range of addresses that correspond to one single line in C source. When you view these breakpoints in C source mode, the graphical representation of the breakpoint dots may be changed to indicate multiple breakpoints for a single line in the C source:

- For a breakpoint within the range for a source line but not at the beginning of the range, the dot is shown in the upper left-hand corner of the gray square.
- For multiple breakpoints corresponding to a single source line, the breakpoints are shown as two smaller, overlapping dots.

TIP: When the source view is the active view, you can use the mouse scroll wheel to step through your code while debugging. Click in the source view to make sure it is the active view.

- Use CTRL+MWHEEL_DOWN to step over
 - Use CTRL+SHIFT+MWHEEL_DOWN to step into
 - Use CTRL+MWHEEL_UP to step out
-

Source View Context Menu

You can use the following functions from the pop-up menu (displayed with the right mouse button) in the source view:

Source

Select **Source** to view the original source code for the current file.

Mixed for File

Select **Mixed for File** to view the original source mixed with the assembly version for the entire address range corresponding to the current source file being displayed.

Note that when you select this option, Palm OS Debugger may take some time to update to the source view for a large source file.

Mixed for Function

Select **Mixed for Function** to view the original source mixed with the assembly version for the current function. The original source is shown for other functions.

Mixed for Source Line

Select **Mixed for Source Line** to view the original source mixed with the assembly version for the current source line only. The original source is shown for all other lines.

Disassembly

Select **Disassembly** to view the disassembly version of the current file.

Note that when you select this option, Palm OS Debugger may take some time to update to the source view for a large source file.

Return to Current PC

Returns you to the location specified by the current program counter data.

Disassemble

Opens a dialog box asking you for the address of the code to disassemble. This function disassembles a raw address range starting from the value you specified in the dialog box. Palm OS Debugger initializes the starting address to the current address range for the file being disassembled.

You specify the disassemble range as a standard ANSI C string in one of the following ways:

- Using a low address and a high address. For example, if you specify the string "0x8000-0x9000", Palm OS Debugger disassembles the address range 0x00008000 to 0x00009000.

- Using a starting address and a size. For example, if you specify the string "0x8000+0x1000", Palm OS Debugger disassembles the address range 0x00008000 to 0x00009000.
- Using a starting address. If you don't specify a size for the disassemble range, then Palm OS Debugger uses the most recently specified size, or uses the default size of 256 bytes if you have never specified a size.

When you click **Enter**, Palm OS Debugger disassembles the entire file up to the upper bound of the disassembler.

Set Disassembler

Changes the disassembler that is being used to disassemble the current source range when in mixed or disassembly view modes. Palm OS Debugger attempts to identify the correct disassembler, but you can set it manually when Palm OS Debugger is unable to determine the correct disassembler.

This function allows you to choose between the disassembler plugins you have installed. Palm OS Debugger ships with 68K, ARM, and Thumb disassemblers.

Set Breakpoint at Address

Sets a breakpoint at an absolute address. This function displays a dialog asking you for the address at which to set the breakpoint. Palm OS Debugger attempts to resolve this address to a source file and line number.

If Palm OS Debugger is able to resolve the address, it stores the breakpoint as a source file and line number breakpoint. If Palm OS Debugger is not able to resolve the address, then the breakpoint is stored as an absolute addressed breakpoint.

Set Breakpoint at Function

Sets a breakpoint at a function. You can either use a dialog to enter the name of the function, or select a function name from a list.

If you use the dialog, you can enter the name of the function in one of two ways:

- As a simple function name. Palm OS Debugger searches the symbolic files sequentially until it finds a matching function name.

Running Palm OS Debugger

Using the Palm OS Debugger Windows

- As a symbolic file and function name combination. For example, if you enter `symfilename.ext(funcName)`, then Palm OS Debugger sets a breakpoint at the function `funcName` in the symbolic file `symfilename.ext`. The symbolic file extension is optional; you can alternatively enter the example above as `symfilename(funcName)`.

If Palm OS Debugger is unable to find the specified function, then it displays an error message.

If you use the function name list, you simply select the function from the list. By default, the function name list is in source file order (the order in which the functions appear in the source file). To sort the function name list in alphabetical order, hold down the shift key before displaying the pop-up menu.

Set Breakpoint at Cursor

Sets a breakpoint in the source view pane at the current cursor location. (This menu item is available when the selection caret is on a source line that has a breakpoint gray box on it.)

Run to Cursor

Sets a temporary breakpoint in the source view pane at the current cursor location, and runs to the cursor location. The breakpoint is cleared at the next stop of program execution.

Set PC at Cursor

Sets the program counter to the current cursor location in the source view pane.

Goto PC

Takes you to the location in source that corresponds with the current contents of the program counter.

Goto Function

Displays a list of function names in the program. Select a function name to move the cursor location in the source view pane to the beginning of that function.

By default, the function name list is in source file order (the order in which the functions appear in the source file). To sort the function name list in alphabetical order, hold down the shift key before displaying the pop-up menu.

Show Line Numbers

For source and mixed modes, displays line numbers in the margin to the left of breakpoint boxes.

Lookup Address

Opens a dialog box asking you for an address that you are looking up. If the address you enter is within the address range for a source file, then that source file is displayed.

Files View

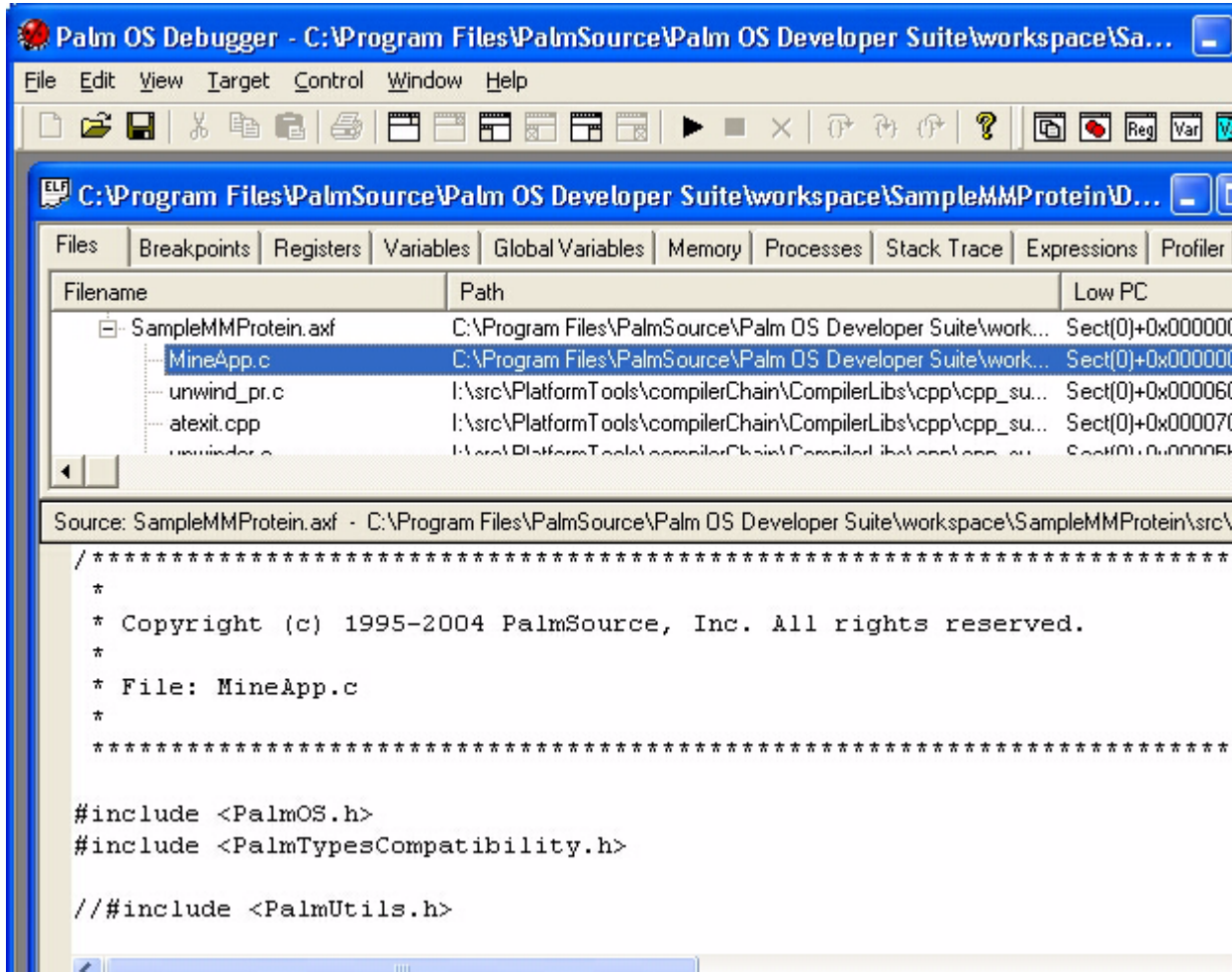
The Files view is the first tab in the user interface, and it is displayed in the upper pane of the user interface. The Files view shows all currently loaded symbolic files.

Expand a symbolic file to display a list of source files that have debugging information. Use the address range information for each symbolic and source file to manually identify which symbolic file and source file an arbitrary address corresponds to.

Running Palm OS Debugger

Using the Palm OS Debugger Windows

Figure 4.2 Files View



Files View User Interface

The files view pane is a resizable pane that displays the filenames, paths, low PC value, and high PC value for the application in a table format. By clicking on one of the column headings, you can sort the table by that value.

When the files view is initially displayed, each symbolic file is collapsed. To expand the contents of a symbolic file, click on the plus sign next to the symbolic file's filename. The contents are the names of the files that have debugging information.

When you click on a filename, the source code for that file is displayed in the [Source View](#).

When you double-click on a filename, Palm OS Debugger tries to open the file in a new window. If Palm OS Debugger cannot find the file, then Palm OS Debugger displays a dialog box so you can locate the source file in another directory.

Files View Context Menu

You can use the following functions from the pop-up menu (displayed with the right mouse button) in the files view:

Open File in New Window

If you have a specific file selected, this option becomes available in the context menu. This option lets you display the file in a separate window.

Remove Symbolic File

To remove a symbolic file from the current debug project, select the symbolic file's filename and then select **Remove Symbolic File** from the pop-up menu. If you set any breakpoints in the symbolic file, Palm OS Debugger removes the breakpoints before removing the symbolic file from the project.

Change Path

Opens a dialog box so you can find the source file in a different directory. If the path of your source file is out of date with the symbolic file, modify the path to your source file by selecting the filename, and then selecting **Change Path** from the pop-up menu. Use this option when a file is located in a different directory but still has the same name and extension.

Change Filename

Opens a dialog box so you can select a different source file. If the path and filename of your source file is out of date with the symbolic file, modify the path and filename to your source file by selecting the filename, and then selecting **Change Filename** from the pop-up menu. Use this option when the file has a different name or extension, and is not just located in a different directory.

Revert to Original Path

Sets the path information back to the path information that is stored in the symbolic file. If you have changed the filename using **Change Filename**, but selected an incorrect source file, you can restore the original name by selecting the filename and then selecting **Revert to Original Path** from the pop-up menu.

Show All Files

By default, the Files view only shows files that contain code. To view other files in the files view, check the **Show All Files** menu. This allows you to see other files which are listed in the symbolic file, such as header files.

Copy

Copies the text from a tree item, and all its available children, to the operating system clipboard. Each tree item is formatted to a separate line with the column text tab-separated.

Copy All

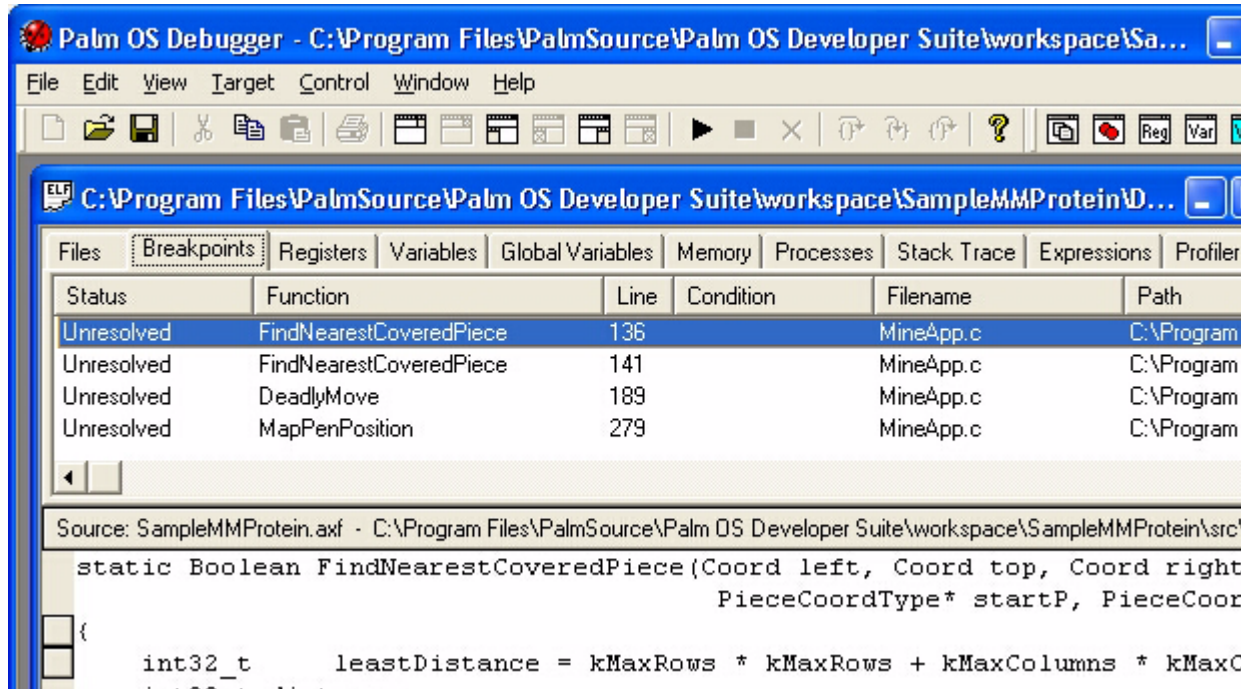
Copies the text for all visible tree items to the operating system clipboard. Each tree item is formatted to a separate line with the column text tab-separated.

Breakpoints View

This view shows the list of breakpoints set in the application you are debugging.

NOTE: You set breakpoints in the source view. For details, see [“Setting Breakpoints in the Source View”](#) on page 50.

Figure 4.3 Breakpoints View



Breakpoints View User Interface

The breakpoints view displays the following information, displayed in a table format. By clicking on any column heading, you can sort the table by that value in either ascending order or descending order. Column headings are:

- *Status* – The current status of the breakpoint. The status is one of the following:
 - *Unresolved*: The breakpoint is set, but the symbolic file containing the breakpoint has not yet been loaded. In the source view, the status for this type of breakpoint is shown with a white dot.
 - *Enabled*: The breakpoint is set and the symbolic file containing the breakpoint has been loaded. The breakpoint reflects an actual address. In the source view, the status for this type of breakpoint is shown with a red dot.

Running Palm OS Debugger

Using the Palm OS Debugger Windows

- Disabled: The breakpoint address is resolved, but the breakpoint is not active. In the source view, the status for this type of breakpoint is shown with a white dot.
- Unknown: If you try to set a breakpoint while a debug process is running, Palm OS Debugger may temporarily set the status in the breakpoints view to Unknown.
- *Function* – The function in which the breakpoint can be found.
- *Line* – The line in the source file.
- *Filename* – The filename of the source file.
- *Path* – The path to the source file.
- *Symbolic File* – The associated symbol file.
- *Condition* – If this is a conditional breakpoint, the condition is displayed here.

Conditional Breakpoints

To add a conditional breakpoint:

- Select the desired line in the breakpoints view. The selected line is highlighted.
- Double-click the section of the highlighted line that runs through the **Condition** column (under the heading “Condition” in the table).
- Palm OS Debugger opens a text box where you add a condition.

When Palm OS Debugger hits your conditional breakpoint, a dialog box opens.

Use Palm OS Debugger’s C expression parser to write conditions. For detailed information, see “[Expressions View](#)” on page 70.

Breakpoints View Context Menu

You can use the following functions from the pop-up menu (displayed with the right mouse button) in the breakpoints view:

Enable Breakpoint

Enables a previously disabled breakpoint at a specific address.

Disable Breakpoint

Disables a breakpoint at a specific address.

Clear Breakpoint

Clears a breakpoint at a specific address.

Enable All Breakpoints

Palm OS Debugger attempts to enable all disabled and unresolved breakpoints. Any unresolved breakpoints that Palm OS Debugger can not set remain unresolved; any disabled breakpoints that Palm OS Debugger can not enable become unresolved.

Disable All Breakpoints

Disables all currently set breakpoints.

Clear All Breakpoints

Clears all currently set breakpoints.

Set Breakpoint at Address

Sets a breakpoint at an absolute address. This function displays a dialog asking you for the address at which to set the breakpoint. Palm OS Debugger attempts to resolve this address to a source file and line number.

If Palm OS Debugger is able to resolve the address, it stores the breakpoint as a source file and line number breakpoint. If Palm OS Debugger is not able to resolve the address, then the breakpoint is stored as an absolute addressed breakpoint.

Set Breakpoint at Function

Sets a breakpoint at a function. This function displays a dialog asking you for the name of the function at which to set the breakpoint. You can enter the name of the function in one of two ways:

- As a simple function name. Palm OS Debugger searches the symbolic files sequentially until it finds a matching function name.
- As a symbolic file and function name combination. For example, if you enter `symfilename.ext(funcName)`, then Palm OS Debugger sets a breakpoint at the function

Running Palm OS Debugger

Using the Palm OS Debugger Windows

`funcName` in the symbolic file `symfilename.ext`. The symbolic file extension is optional; you can alternatively enter the example above as `symfilename(funcName)`.

If Palm OS Debugger is unable to find the specified function, then it displays an error message.

Copy

Copies the text from a tree item, and all its available children, to the operating system clipboard. Each tree item is formatted to a separate line with the column text tab-separated.

Copy All

Copies the text for all visible tree items to the operating system clipboard. Each tree item is formatted to a separate line with the column text tab-separated.

Registers View

This view displays the list of registers and their values. It has bitwise text mnemonics for registers such as the CPSR and SPSR, to mimic the ARM debugger.

Registers View User Interface

You can sort the view by the name and by the value, in both ascending or descending order.

To set the contents of a register, you can double-click on the value of a register, and type in a new value. This value gets reflected immediately in the debugging session.

In general, Palm OS Debugger interprets the value according to the semantics of the ANSI C standard function `strtol` (or `strtod` for floating point numbers).

- If the value starts with a number (1 through 9) or with a hyphen (“-”), then Palm OS Debugger interprets the value as a decimal number.
- If the value starts with “0x” characters, then Palm OS Debugger interprets the value as a hexadecimal number.

- If the value starts with “0” followed by the numbers 1 through 7, then Palm OS Debugger interprets the value as an octal number.
- If the value contains a decimal point (“.”), then Palm OS Debugger interprets the value as one of the IEEE 754 (floating point number) formats, depending on the size of the register.
- If the value starts with “0b” characters followed by zeroes or ones, then Palm OS Debugger interprets the value as a binary number.

You can enter any valid numerical string; you do not need to use quotation marks. If the string cannot be parsed according to the above rules, Palm OS Debugger interprets it based on the format of the previous value.

For example, say that a register's value is "0x11223344" and you change its value to "be": Palm OS Debugger attempts to parse the value based on the rules described above. But since none of the rules fit, Palm OS Debugger then parses the string in hexadecimal format because it was originally in hexadecimal format.

If Palm OS Debugger is unable to determine the correct format of the string, then the register's value remains unchanged. If you specify an out-of-range string, the value also remains unchanged.

Registers View Context Menu

You can use the following functions from the pop-up menu (displayed with the right mouse button) in the register view:

NOTE: If an individual register is selected, then options selected in the context menu apply only to that register. If a group of registers is selected, then options selected in the context menu apply to all registers in that group.

Format Types

You can display registers in the following formats:

- ASCII Integer
- Binary
- Boolean

Running Palm OS Debugger

Using the Palm OS Debugger Windows

- Char
- Decimal (Signed)
- Decimal (Unsigned)
- Fixed 50/50 (Signed)
- Fixed 50/50 (Unsigned)
- Fixed 75/25 (Signed)
- Fixed 75/25 (Unsigned)
- Float
- Hex
- Hex Byte Stream
- Hex with Decimal (Signed)
- Hex with Decimal (Unsigned)
- String (C)
- String (Pascal)

Format Changes Are Global

Select this setting to permanently apply changes to a register for the duration of the debug session. If you create any new register views, they have the same format.

Copy

Copies the text from a tree item, and all its available children, to the operating system clipboard. Each tree item is formatted to a separate line with the column text tab-separated.

Copy All

Copies the text for all visible tree items to the operating system clipboard. Each tree item is formatted to a separate line with the column text tab-separated.

Variables View

This view displays information about variables: the variables' names, the values of the variables, the types of the variables, the scope of the variables, and the variables' locations. If a variable is

out of scope, then Palm OS Debugger displays the string "ERROR" for that variable.

Variables View User Interface

You can sort the table of data in ascending or descending order by clicking on any of the column titles. To change the value of a variable, double-click on the value of a variable and type in a new value. The new value gets reflected immediately in the debugging session.

- If the value starts with a number (1 through 9) or with a hyphen ("-"), then Palm OS Debugger interprets the value as a decimal number.
- If the value starts with "0x" characters, then Palm OS Debugger interprets the value as a hexadecimal number.
- If the value starts with "0" followed by the numbers 1 through 7, then Palm OS Debugger interprets the value as an octal number.
- If the value contains a decimal point ("."), then Palm OS Debugger interprets the value as an IEEE floating point number.
- If the value starts with "0b" characters followed by zeroes or ones, then Palm OS Debugger interprets the value as a binary number.

You can enter any valid numerical string; you do not need to use quotation marks. If the string cannot be parsed according to the above rules, Palm OS Debugger interprets it based on the format of the previous value.

For example, say that a variable's value is "0x11223344" and you change its value to "be": Palm OS Debugger attempts to parse the value based on the rules described above. But since none of the rules fit, Palm OS Debugger then parses the string in hexadecimal format because it was originally in hexadecimal format.

If Palm OS Debugger is unable to determine the correct format of the string, then the variable's value remains unchanged. If you specify an out-of-range string, the value also remains unchanged.

Variables View Context Menu

You can use the following functions from the pop-up menu (displayed with the right mouse button) in the variables view:

Show File Globals

When this command is checked, the file global variables are shown.

Format Types

You can display variables in various formats, similar to the way you can display registers in various formats (see “[Format Types](#)” on page 63). Additionally, you can show enumerated types as their enumerated strings.

Format Elements As

When you select an array name, you can use the **Format Elements As** command. This command displays all elements of an array in the same format. You can display the arrays of characters as strings.

Copy

Copies the text from a tree item, and all its available children, to the operating system clipboard. Each tree item is formatted to a separate line with the column text tab-separated.

Copy All

Copies the text for all visible tree items to the operating system clipboard. Each tree item is formatted to a separate line with the column text tab-separated.

Global Variables View

This view displays the global variable name, the value of the variables, the type of variable, the variable location.

Global Variables User Interface

You can sort the view by clicking on any of the titles to sort in ascending or descending order. Additionally, you can double click on the value of a variable and type in a new value. The new value gets reflected immediately in the debugging session.

You can edit either the value or location of global variables in the same way that you can edit variables (see [Variables View](#)).

Global Variables Context Menu

You can use the following functions from the pop-up menu (displayed with the right mouse button) in the global variables view:

Show All Globals

When this command is checked, all global variables are shown.

Format Types

You can display variables in various formats, similar to the way you can display registers in various formats (see “[Format Types](#)” on page 63). Additionally, you can show enumerated types as their enumerated strings.

Format Elements As

When you select an array name, you can use the **Format Elements As** command. This command displays all elements of an array in the same format. You can display the arrays of characters as strings.

Copy

Copies the text from a tree item, and all its available children, to the operating system clipboard. Each tree item is formatted to a separate line with the column text tab-separated.

Copy All

Copies the text for all visible tree items to the operating system clipboard. Each tree item is formatted to a separate line with the column text tab-separated.

Memory View

This view displays memory address locations and their contents (memory dumps). You can display memory either using the built-in memory display function or using any memory disassembler plug-in.

You specify the absolute start memory address as a standard ANSI C string representing a hexadecimal number (for example, “0x8000”).

Memory View User Interface

- You can sort the view by clicking on any of the titles to sort in ascending or descending order, where applicable.
- By double-clicking on a specific address, range of data, or displayed character string, you can type in a new value and press ENTER to save the changed value.

Memory View Context Menu

You can use the following functions from the pop-up menu (displayed with the right mouse button) in the memory view:

Format

Sets the data type format to be used for the displayed data.

Bytes per Column

Sets the number of bytes of data to be displayed in each column.

Bytes per Row

Sets the number of bytes of data to be displayed in each row.

Uppercase Numbers

Select this setting to show hexadecimal numbers with uppercase letters. For example, with this setting, the number 0xabcdef would be displayed as 0xABCDEF.

Show Base Prefix

Select this setting to display the base prefix for the number being displayed. This setting shows the “0x” prefix in front of hexadecimal numbers and the “0b” prefix in front of binary numbers. For example, the hexadecimal number 1122aabb displays as 0x1122aabb if this option is selected, but displays as 1122aabb if this option is not selected.

68K Disassembler

Displays the 68K Disassembler’s various modes. Selecting one disassembles the selected address.

ARM Disassembler

Displays the ARM Disassembler’s various modes. Selecting one disassembles the selected address.

Number Disassembler

Displays the various modes that the Number disassembler has.

Thumb Disassembler

Displays the various modes that the Thumb disassembler has.
Selecting one disassembles the selected address.

Copy

Copies the text from a tree item, and all its available children, to the operating system clipboard. Each tree item is formatted to a separate line with the column text tab-separated.

Copy All

Copies the text for all visible tree items to the operating system clipboard. Each tree item is formatted to a separate line with the column text tab-separated.

Processes View

This view displays processes that Palm OS Debugger knows about. For example, Palm OS Debugger notifies you when a thread faults or has an uncaught exception. To view the faulted thread, double-click on the thread in the Processes view. The thread name is displayed next to the thread ID (in Palm OS Cobalt, this is the four-character code for the thread). Double-clicking on a thread tree item entry shows the thread's context in the Source view window and updates all views.

Processes View User Interface

You can sort the view by clicking on any of the titles to sort in ascending or descending order. You can sort by Name or Value.

Processes View Context Menu

You can use the following functions from the pop-up menu (displayed with the right mouse button) in the processes view:

Remove All Added Columns

Removes all added columns.

Copy

Copies selected text in the current window into the operating system clipboard.

Copy All

Copies all text in the current window into the operating system clipboard.

Stack Trace View

This view displays the stack, allowing you to see the previous calls.

Stack Trace View User Interface

You cannot sort the stack trace view.

Stack Trace View Context Menu

You can use the following functions from the pop-up menu (displayed with the right mouse button) in the stack trace view:

Copy

Copies the text from a tree item, and all its available children, to the operating system clipboard. Each tree item is formatted to a separate line with the column text tab-separated.

Copy All

Copies the text for all visible tree items to the operating system clipboard. Each tree item is formatted to a separate line with the column text tab-separated.

Expressions View

This displays a text window where you can type a C expression and have it evaluated. You can type in an expression and Palm OS Debugger evaluates it and displays the result. The expression can contain simple variables (no array members and no struct, union, or class members), registers, memory dereferences, and constants.

The expression parser supports the following operators with full C operator precedence (topmost is highest precedence).

Table 4.1 Operators Supported by the C Expression Parser

Expression	How It Is Read
()	left to right
! ~ ++ --	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
	left to right
&&	left to right
	left to right
?:	right to left
= += -= *= /= %= &= ^= = <<= >>=	right to left

All operations are supported for SInt32, UInt32, SInt64, UInt64, and 4-byte floats. Constant numbers can be in expressions and are entered as they would be in a C program. Integers are SInt32 by default. Integers can be unsigned by following them with a U or UL (for example, 3u). 64-bit integers can be used in expressions by typing a hex number that is too large for a 32-bit value (for example, 0x000000001). Floats can be entered in any format and are assumed to be 4-byte floats by default.

Expressions can currently access debug variables by typing the variable name in the expression. The expression parser can use

simple variables and does not support members of structs, unions or classes. Registers can be accessed as @<reg_name> (for example, @R0).

The register name is case sensitive. The register name must match the exact name that is found in the register views for it to evaluate correctly. In addition, the register must be valid at the time of the expression being evaluated; the parser cannot read registers prior to connecting.

The expression parser also has built-in expression variables that can be defined and are global to all of Palm OS Debugger. These variables are defined as C-tokens pre-pended with a \$ character (for example, \$x); note the similarity to Perl variables.

Table 4.2 Sample Expressions

Expression	What It Does
2 << 20	Simple constant expression
x == 3	Find out if the debug variable x equals 0x00000003
myFloat == 2.0	Find out if myFloat is 2.0 (0x40000000)
\$foo = 1024	Set expression glob var to 1024
\$bar = ++\$foo	Set \$bar to ++ of \$foo
\$foo	print the value of \$foo
@R0	get the value of the register R0
(@CPSR & 0x1FUL) == 0x10UL	Find out if we are in user mode
(@CPSR & (1 << 5))	Find out if we are in Thumb mode

Expressions Console User Interface

You can sort the view by clicking on any of the titles to sort in ascending or descending order. You can sort by Expression, Result, Hex, or Type.

Expressions Context Menu

You can use the following function from the pop-up menu (displayed with the right mouse button) in the expressions view:

New Expression

Provides an input area (in the Expression column) where you can type an expression. The expression parser evaluates the expression and displays the result in the Result column.

Delete Expression

Deletes the selected expression.

Copy

Copies the text from a tree item, and all its available children, to the operating system clipboard. Each tree item is formatted to a separate line with the column text tab-separated.

Copy All

Copies the text for all visible tree items to the operating system clipboard. Each tree item is formatted to a separate line with the column text tab-separated.

Profiler View

This view is not used for application development.

STDIO Console

The STDIO Console is a read-only text window where your application can conduct standard input and output operations. The STDIO Console is typically used for output of debug messages.

To write your program so that it uses the STDIO Console, use the `AdnDebugMessage()` and `AdnDebugEnableSet()` functions described in [Chapter 6](#), “[AdnDebug Manager](#),” on page 95.

The STDIO Console window is colored gray to indicate that the window is read-only. If an input operation such as `scanf` is necessary, the window temporarily turns white to indicate input is necessary to continue execution.

STDIO Console User Interface

In this pane, you can use the following functions:

- Find and Find Next
- Find Previous
- Select and Copy text to the clipboard

STDIO Context Menu

You can use the following functions from the pop-up menu (displayed with the right mouse button) in the STDIO view:

Copy

Copies the text from a tree item, and all its available children, to the operating system clipboard. Each tree item is formatted to a separate line with the column text tab-separated.

Copy All

Copies the text for all visible tree items to the operating system clipboard. Each tree item is formatted to a separate line with the column text tab-separated.

Select All

Selects all of the text in the current window.

Clear

Clears the current console.

Echo Input

Select this setting to indicate that the input typed should appear in the console as text.

Sync on EOL

Select this setting to force the console buffer to flush itself when it detects an end-of-line character. This setting allows you to see console information more quickly.

Debug Console

The debug console is a command line interface, allowing you to interact with your debugging session in a command line environment.

Debug Console User Interface

The user interface of the debug console is a command line interface. The percentage sign, %, is the command line prompt.

Debug Console Commands

Command	Arguments
Alias	[<NewAlias> <actualCommand>] This allows you to make a new alias of an actual command. Typing this command without any parameters displays all currently assigned aliases.
Bpclear	[<address> <functionname> <symbolname> <symfile(functionname)>] Clears a breakpoint at an address, function, or symbol.
Bpset	[<address> <functionname> <symbolname> <symfile(functionname)>] Sets a breakpoint at an address, function, or a symbol.
Clear	Clears the console contents.
Find	[<function name> <symbol name> symfile(function name)> <symfile(symbol name)>] Finds a function or symbol address and size from symbolics.
Help	[<command name>]

	Displays help for all commands or for a specific command.
Lookup	<p><absolute address></p> <p>Lets you look up an address in symbolics and see to which symbolic file, source file and source line, function, and scope the address pertains.</p>
Mem	<p>[<absolute address> [numbytes=32]]</p> <p>Displays memory at an absolute address.</p>
Prot	<p>Sends commands to the debugger protocol.</p> <p>To find out what protocol interface commands are supported, use this command: prot help</p>
Prot nub	<p>Sends commands to the debugger protocol nub.</p> <p>To find out what protocol nub commands are supported, use this command: prot nub help</p>
Rename	<p><old command name> <new command name></p> <p>Renames a command.</p>
Setsym	<p><sym file name></p> <p>Sets the current symbolic file for 'sym' commands</p>
sym	<p><varies based on symbolic plug-in interface></p> <p>Sends a command down to the current symbolic file's symbolic plug-in interface</p>

Debug Console Context Menu

You can use the following functions from the pop-up menu (displayed with the right mouse button) in the debug console view:

Copy

Copies selected text in the current window into the operating system clipboard.

Paste

Pastes the contents of the operating system clipboard into the current window.

Select All

Selects all of the text in the current window.

Clear

Clears the current console

Echo Input

Select this setting to indicate that the input typed should appear in the console as text.

Sync on EOL

Select this setting to force the console buffer to flush itself when it detects an end-of-line character. This setting allows you to see console information more quickly.

Running Palm OS Debugger

Using the Palm OS Debugger Windows

Palm OS Debugger Menu Reference

This chapter provides reference information describing each menu command in Palm OS Debugger.

Palm OS Debugger Menu Reference Overview

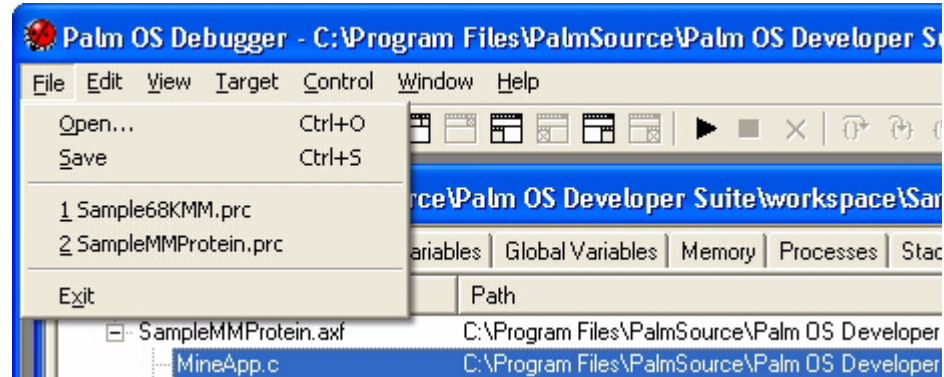
The Palm OS Debugger menus include:

- “[File](#)” on page 79
- “[Edit](#)” on page 81
- “[View](#)” on page 84
- “[Target](#)” on page 85
- “[Control](#)” on page 88
- “[Window](#)” on page 89
- “[Help](#)” on page 92

File

Use the **File** menu, shown in [Figure 5.1](#), to perform actions related to the current file you are debugging.

Figure 5.1 File Menu



Open

Use the **Open** menu to open a new Palm OS Debugger window with an existing target file.

Save

Use the **Save** menu to save the current window's contents to a file. This menu item cannot be used with a debug target file (PRC or ELF file).

Recent File List

Palm OS Debugger adds each open debug target to the **File** menu, in chronological order, so that you can select recently used files when starting a new debug session.

The list shows the short filenames, but the path to each file is displayed in the message area at the bottom of the Palm OS Debugger window.

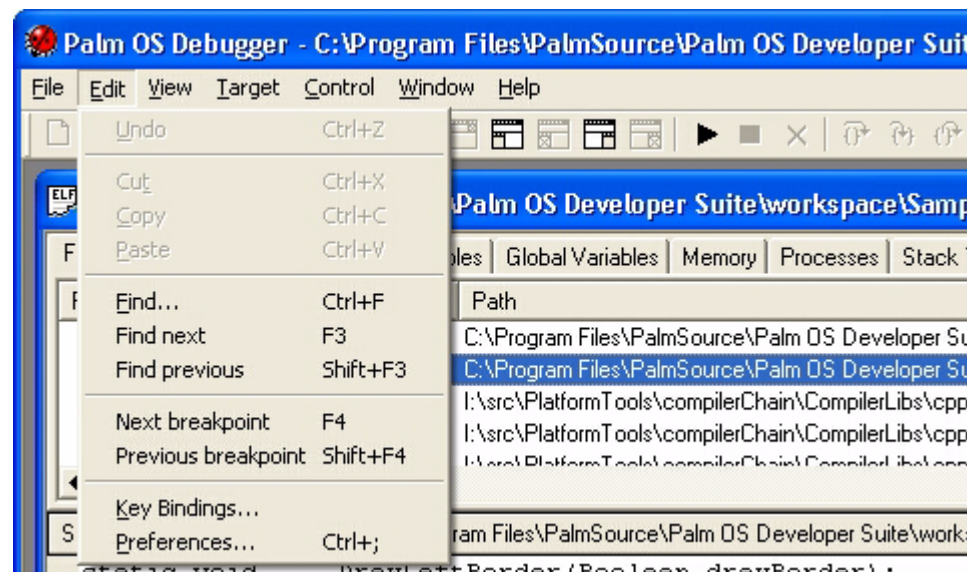
Exit

Use the **Exit** menu to exit Palm OS Debugger.

Edit

Use the **Edit** menu, shown in [Figure 5.2](#), to perform standard editing functions, to search for text strings, to move through the breakpoints, and to change keyboard settings and debugger preferences.

Figure 5.2 Edit Menu



Undo

Use the **Undo** menu to undo the previously executed edit command.

Cut

Use the **Cut** menu to delete the currently selected text, copying the selected text into the operating system clipboard.

Copy

Use the **Copy** menu to copy selected text in the current window into the operating system clipboard.

Paste

Use the **Paste** menu to copy text from the clipboard into the current edit window.

Find

Use the **Find** menu to search for a text string in a window.

Find next

Use the **Find next** menu to look for a text string in a window using an advance search direction (searching forward from the current position).

Find previous

Use the **Find previous** menu to look for a text string in a window using a reverse direction search (searching backwards from the current position).

Next breakpoint

Use the **Next breakpoint** to jump to the next code line where you can place a breakpoint in the source window.

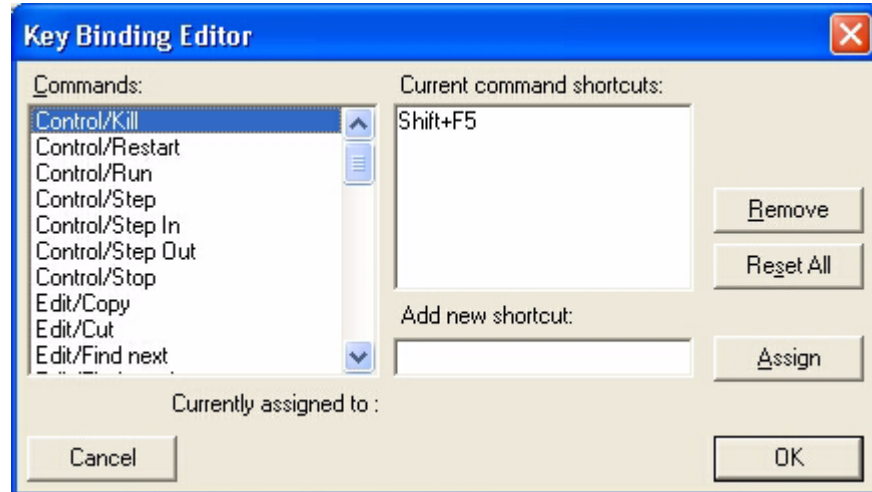
Previous breakpoint

Use the **Previous breakpoint** menu to jump back to the next prior code line where you can place a breakpoint in the source window.

Key Bindings

Use the **Key Bindings** menu to display the Key Bindings Editor, shown in [Figure 5.3](#).

Figure 5.3 Key Bindings Editor Dialog



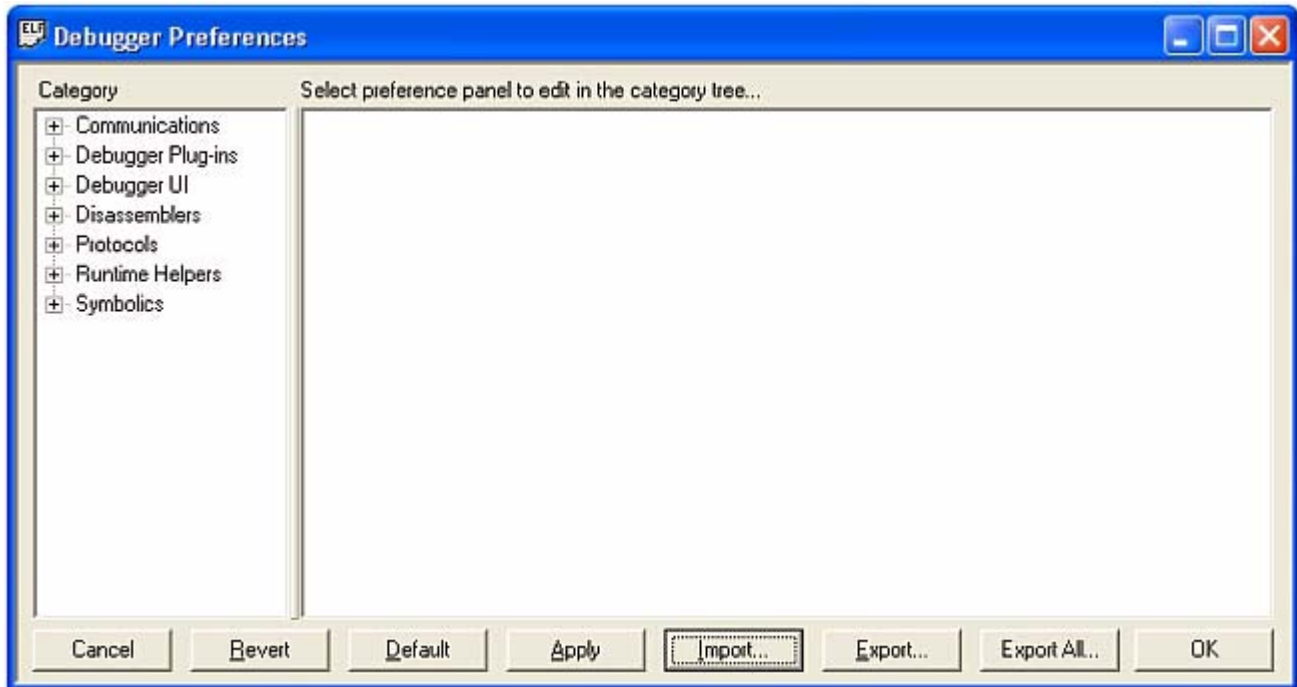
Preferences

Use the **Preferences** menu to display the Debugger Preferences dialog box, shown in [Figure 5.4](#).

Palm OS Debugger Menu Reference

View

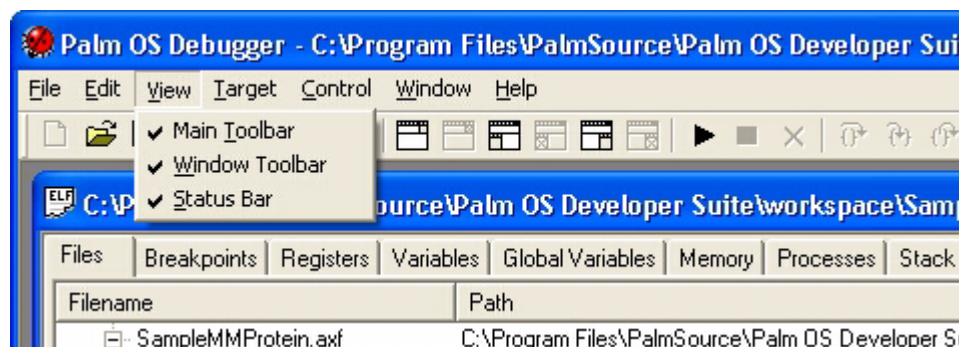
Figure 5.4 Debugger Preferences Dialog Box



View

Use the **View** menu, shown in [Figure 5.5](#), to indicate whether Palm OS Debugger should show the toolbars or the status bar.

Figure 5.5 View Menu



Main Toolbar

Use the **Main Toolbar** menu to hide or show the icons in the toolbar at the top of the main window.

Window Toolbar

Use the **Window Toolbar** menu to hide or show the toolbar icons for the **Window** items.

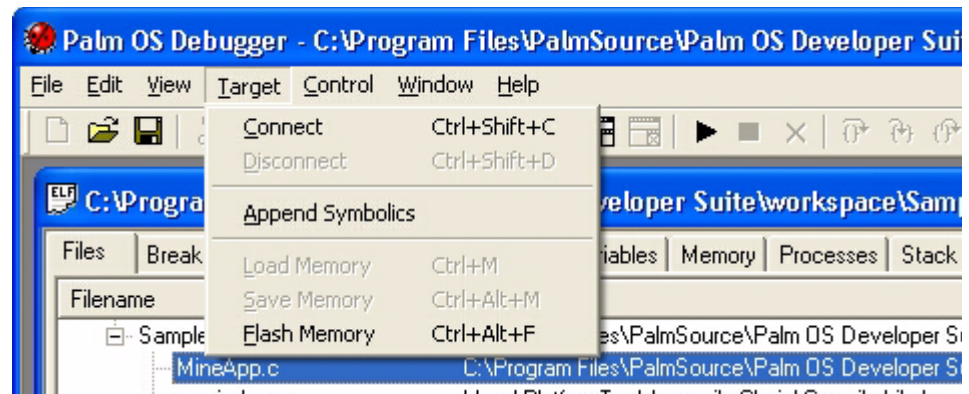
Status Bar

Use the **Status Bar** menu to hide or show the status information at the bottom of the main window.

Target

Use the **Target** menu, shown in [Figure 5.6](#), to connect and disconnect from a debug target, to use a symbolics file, or to save and load memory values.

Figure 5.6 Target Menu



Connect

Use the **Connect** menu to connect the Palm OS Debugger with a debug target.

Disconnect

Use the **Disconnect** menu to disconnect the Palm OS Debugger from a debug target.

Append Symbolics

A *symbolics file* is a file containing information that Palm OS Debugger needs in order to display the source code that corresponds to your object code.

Use the **Append Symbolics** menu to select a symbolics file to append. Palm OS Debugger loads the code symbolics without requiring the code to be downloaded to the debug target.

You can use the symbolics to set breakpoints in the Files view. Any breakpoints that you set prior to connecting to the debug target and running the first time are set as unresolved. When you connect to the debug target, Palm OS Debugger attempts to resolve these breakpoints.

Remove Symbolics

Use the **Remove Symbolics** menu to remove the symbolics that had been previously added using the **Append Symbolics** menu.

Load Memory

Use the **Load Memory** menu to load the memory contents from a previously saved file. Select the binary file to load, and enter the absolute memory location to be loaded.

You specify the memory location range as a standard ANSI C string in one of the following ways:

- Using a low address and a high address. For example, if you specify the string "0x8000-0x9000", Palm OS Debugger loads the address range 0x00008000 to 0x00009000.
- Using a starting address and a size. For example, if you specify the string "0x8000+0x1000", Palm OS Debugger loads the address range 0x00008000 to 0x00009000.

- Using a starting address. If you don't specify a size for the memory range, then Palm OS Debugger loads the entire binary file into the memory starting at the starting address.

Palm OS Debugger downloads the file contents as a binary file into the specified memory location.

NOTE: The examples above show the address values and size specified as hexadecimal numbers. The address values and size can be specified in decimal, hexadecimal, octal, floating point, or binary, as described in "[Registers View User Interface](#)" on page 62.

Save Memory

Use the **Save Memory** menu to save the current memory contents to a file. Enter a file name for the memory contents, and the absolute memory location to be saved.

You specify the memory location range as a standard ANSI C string in one of two ways:

- Using a low address and a high address. For example, if you specify the string "0x8000-0x9000", Palm OS Debugger saves the address range 0x00008000 to 0x00009000.
- Using a starting address and a size. For example, if you specify the string "0x8000+0x1000", Palm OS Debugger saves the address range 0x00008000 to 0x00009000.

If you specify a starting address and no size, then Palm OS Debugger saves an empty file (a file of size 0).

NOTE: The examples above show the address values and size specified as hexadecimal numbers. The address values and size can be specified in decimal, hexadecimal, octal, floating point, or binary, as described in "[Registers View User Interface](#)" on page 62.

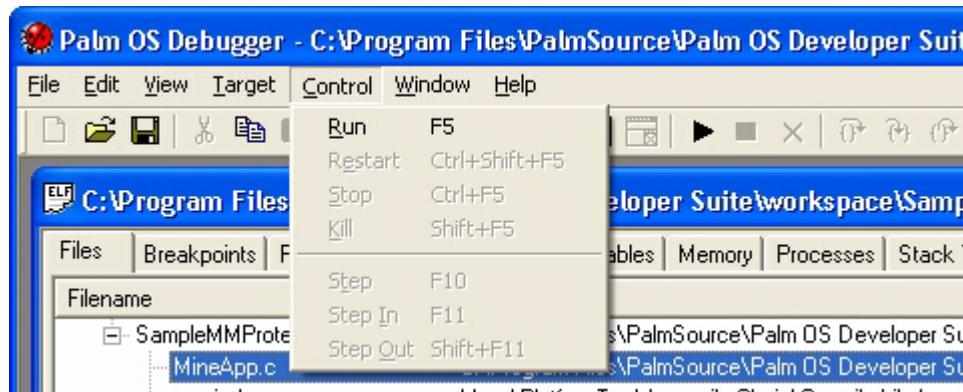
Flash Memory

This menu is intended for licensee use.

Control

Use the **Control** menu, as shown in [Figure 5.7](#), to control the execution of the program you are debugging.

Figure 5.7 Control Menu



Run

Use the **Run** menu to execute a program on the debug target, stopping only at breakpoints or at the normal end of the program execution.

If you have not yet connected to a debug target, Palm OS Debugger tries to connect. If the connection is successful, then Palm OS Debugger downloads the code to the debug target and runs it.

If you have already connected to the debug target, Palm OS Debugger runs the code that was already downloaded.

Restart

Use the **Restart** menu to go back to the beginning of a program, restarting the debugging session.

Stop

Use the **Stop** menu to suspend program execution during a debugging session.

Kill

Use the **Kill** menu to completely stop the execution of the program you are debugging and end the debugging session. This menu item will also disconnect Palm OS Debugger from the debug target.

Step

Use the **Step** menu to execute code, a single source line at a time, in the program being debugged.

Step In

Use the **Step In** menu to execute code a single source line at a time, stepping into a subroutine if execution branches into a subroutine.

Step Out

Use the **Step Out** menu to execute the rest of the code in a subroutine, stopping if execution steps out of the subroutine.

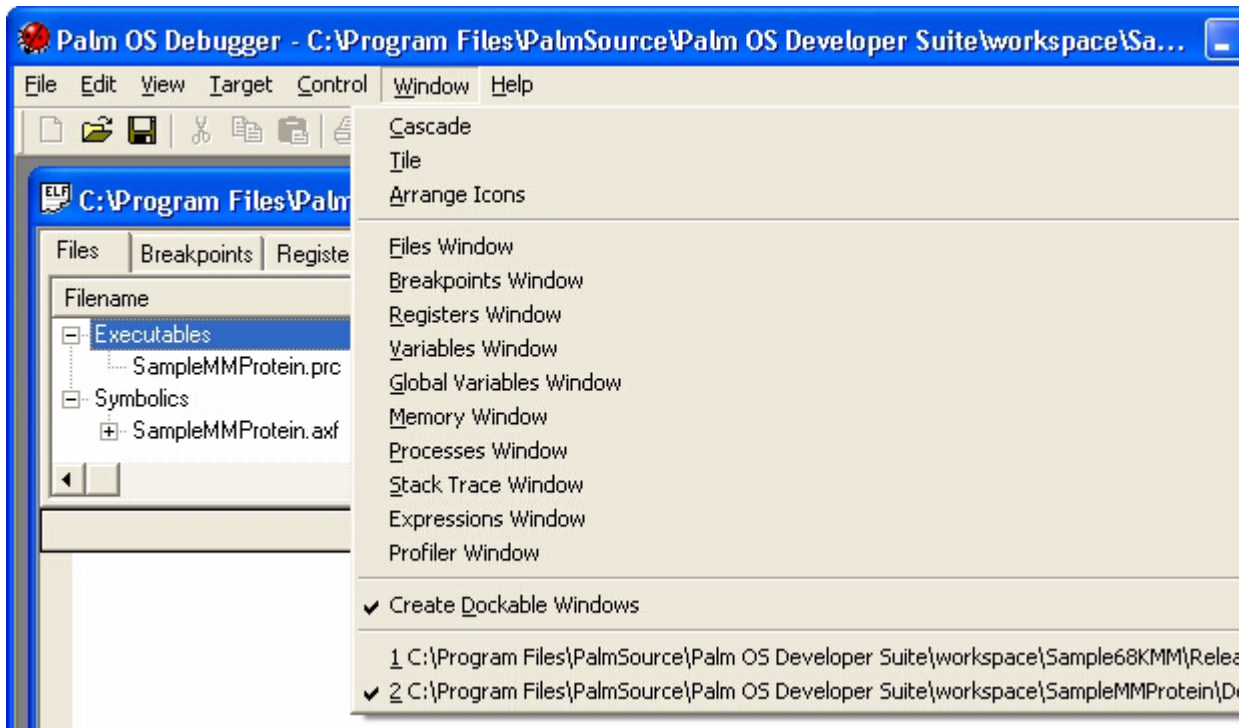
Window

Use the **Window** menu, shown in [Figure 5.8](#), to open new debugging windows and to arrange the open debug windows.

Palm OS Debugger Menu Reference

Window

Figure 5.8 Window Menu



Cascade

Use the **Cascade** menu to have the open windows displayed in an overlapping arrangement, with the title bar on each window visible.

Tile

Use the **Tile** menu to have the open windows displayed in the main window so that none of the windows overlap.

Arrange Icons

Use the **Arrange Icons** menu to arrange minimized window icons in the main window.

Files Window

Opens the content of the files view in a separate window. See "[Files View](#)" on page 55 for information on the files view.

Breakpoints Window

Opens the content of the breakpoints view in a separate window. See “[Breakpoints View](#)” on page 58 for information on the breakpoints view.

Registers Window

Opens the content of the registers view in a separate window. See “[Registers View](#)” on page 62 for information on the registers view.

Variables Window

Opens the content of the variables view in a separate window. See “[Variables View](#)” on page 64 for information on the variables view.

Global Variables Window

Opens the content of the global variables view in a separate window. See “[Global Variables View](#)” on page 66 for information on the global variables view.

Memory Window

Opens the content of the memory view in a separate window. See “[Memory View](#)” on page 67 for information on the memory view.

Processes Window

Opens the content of the processes view in a separate window. See “[Processes View](#)” on page 69 for information on the processes view.

Stack Trace Window

Opens the content of the stack trace view in a separate window. See “[Stack Trace View](#)” on page 70 for information on the stack trace view.

Expressions Window

Opens the content of the expressions view in a separate window. See [“Expressions View”](#) on page 70 for information on the expressions view.

Profiler Window

Opens the content of the profiler view in a separate window. See [“Profiler View”](#) on page 73 for information on the profiler view.

Create Dockable Windows

Select the **Create Dockable Windows** menu to cause the child windows to appear as docked windows attached to the main window. If this menu item is not selected, then child windows are separate child windows.

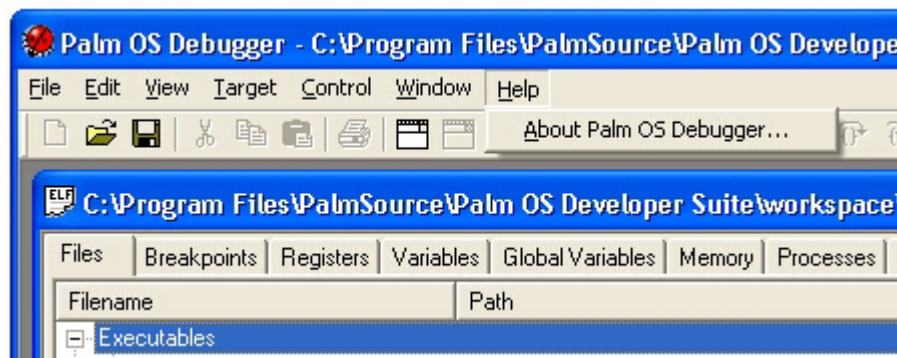
File Names

Palm OS Debugger adds each open debug target to the **Window** menu, so that you can switch between the currently opened windows.

Help

Use the **Help** menu, as displayed in [Figure 5.9](#), to display information about Palm OS Debugger.

Figure 5.9 Help Menu



About Palm OS Debugger

Use the **About Palm OS Debugger** menu to display an About dialog box for Palm OS Debugger.

AdnDebug Manager

This chapter provides documentation for the ARM-based debugger nub manager, called AdnDebug Manager, and is divided into the following sections:

AdnDebug Manager Concepts 96
AdnDebug Manager Constants 97
AdnDebug Manager Functions and Macros 98

The AdnDebug Manager API is declared in the header file `AdnDebugMgr.h`. These functions can be used within ARM subroutines, also called PACE native objects.

AdnDebug Manager Concepts

In order to debug your ARM-based code, your application needs to do the following:

- “[Activate the ARM Debugger Nub](#)” on page 96
- “[Register with Palm OS Debugger](#)” on page 96

Activate the ARM Debugger Nub

The ARM debugger nub is not active on an ARM-based device until an application activates it. Your ARM subroutine must first enable the ARM debugger nub by calling the [AdnDebugEnableSet\(\)](#) macro.

```
// Tell the debugger we want to enable full debugging
UInt32 flags = AdnDebugEnableGet();
flags |= kAdnEnableMasterSwitch | kAdnEnableFullDebugging;
AdnDebugEnableSet(flags);
```

Register with Palm OS Debugger

Palm OS Debugger needs to be able to resolve where each ARM subroutine is located in memory. Your ARM subroutine must register with Palm OS Debugger by calling the [AdnDebugNativeRegister\(\)](#) macro. You need to call the [AdnDebugNativeRegister\(\)](#) macro once after the code resource has been locked, typically on the first call into the ARM subroutine.

However, your code should also unregister (prior to the being unlocked) by calling the [AdnDebugNativeUnregister\(\)](#) macro.

These calls do not act as breakpoints and they do not halt device operation. But they may cause breakpoints to become resolved or unresolved.

```
// Tell the debugger where our code lives in memory:
AdnDebugNativeRegister(
    sysFileTApplication, appFileCreator,
    NativeResourceType, NativeResourceID);
```

The ARM subroutine needs to define `appFileCreator`, `NativeResourceType`, and `NativeResourceID` in order for Palm OS Debugger to locate the corresponding debug symbol file.

For example, if you have an ARMC resource with ID 0001, the debug symbol file is typically of the form `ARMC0001.bin.elf`. The ELF file must be in the same directory as the PRC file that contains the ARM subroutine you are debugging.

AdnDebug Manager Constants

Debugging Constants

Purpose	These constants define debugging features that you can enable and disable.
Declared In	<code>AdnDebugMgr.h</code>
Constants	<pre>#define kAdnEnableDebugIndicator 0x00000002</pre> <p>The debug indicator enables a visual indicator that shows when the debugger nub is performing or waiting for serial communication. For example, on the Tungsten T device, the green LED blinks to indicate that the debugger nub is active.</p> <pre>#define kAdnEnableFullDebugging 0x00000004</pre> <p>The full debugging feature must be enabled in order for application calls to AdnDebugNativeRegister() to be processed correctly. If this feature is disabled, then other native debugging calls (such as <code>ModulePostLoad</code> and <code>ModulePostUnload</code>) are also disabled.</p> <pre>#define kAdnEnableMasterSwitch 0x00000001</pre> <p>When the master switch is enabled, the debug nub catches fatal ARM exceptions, enters the debugger if AdnDebugBreak() is called, and catches <code>ErrFatalDisplay()</code> calls.</p> <p>When the master switch is disabled, the you are limited to calling the AdnDebugEnableSet() and AdnDebugEnableGet() macros.</p>

AdnDebug Manager

AdnDebug Manager Functions and Macros

```
#define kAdnEnableShowSafeFatalAlerts 0x00000008
    When show safe fatal alerts is enabled, the debugger nub
    does not catch calls to ErrFatalDisplay() if the Reset,
    Debug, and Continue options are displayed.
```

AdnDebug Manager Functions and Macros

AdnDebugBreak Macro

Purpose	Break into Palm OS Debugger from the ARM subroutine.
Declared In	AdnDebugMgr.h
Prototype	<pre>#define AdnDebugBreak ()</pre>
Parameters	None.
Returns	Nothing.
Comments	This macro is the ARM-based equivalent to the Debug Manager call DbgBreak().

AdnDebugEnabledGet Macro

Purpose	This macro returns flags indicating which debugging features are enabled.
Declared In	AdnDebugMgr.h
Prototype	<pre>#define AdnDebugEnabledGet ()</pre>
Parameters	None.
Returns	UInt32 representing flags as defined in “ Debugging Constants ” on page 97.
Example	<pre>UInt32 flags = AdnDebugEnabledGet(); flags = kAdnEnableMasterSwitch kAdnEnableFullDebugging; AdnDebugEnabledSet(flags)</pre>
See Also	AdnDebugEnabledSet

AdnDebugEnableGetSupported Macro

Purpose	This macro returns flags indicating which debugging features are supported.
Declared In	<code>AdnDebugMgr.h</code>
Prototype	<code>#define AdnDebugEnableGetSupported ()</code>
Parameters	None.
Returns	UInt32 made up of <code>kAdnEnable</code> flags as defined in “ Debugging Constants ” on page 97.

AdnDebugEnableSet Macro

Purpose	This macro enables the debugging features indicated by the parameter <i>flags</i> .
Declared In	<code>AdnDebugMgr.h</code>
Prototype	<code>#define AdnDebugEnableSet (UInt32 flags)</code>
Parameters	→ <i>flags</i> <code>kAdnEnable</code> flags as defined in “ Debugging Constants ” on page 97
Returns	Nothing.
Comments	The ARM debugger nub is not active on an ARM-based device until an application activates it. The ARM subroutine must first enable the ARM debugger nub by calling the macro <code>AdnDebugEnableSet()</code> .
See Also	AdnDebugEnableGet

AdnDebugLicenseeSpecific Macro

Purpose	This macro makes a licensee-specific call to AdnDebug Manager.
Declared In	<code>AdnDebugMgr.h</code>
Prototype	<code>#define AdnDebugLicenseeSpecific (oemID, selector, param)</code>
Parameters	→ <i>oemID</i> PalmSource-registered OEM ID (creator ID)

AdnDebug Manager

AdnDebugMessage

→ *selector*

Licensee-specific function selector.

→ *param*

Function-specific parameter.

Returns Returns -1 if the licensee-specific function is not supported.

AdnDebugMessage Macro

Purpose This macro displays a debug message in the debugger.

Declared In `AdnDebugMgr.h`

Prototype `#define AdnDebugMessage (messageP)`

Parameters → *messageP*

A pointer to a sequence of ASCII characters terminated by a null character (the string that is displayed in the debugger).

Returns Nothing.

Comments This macro is the ARM-based equivalent to the Debug Manager call `DbgMessage()`.

AdnDebugMessageIf Macro

Purpose This macro displays a debug message in the debugger if the condition specified by the *condition* parameter is true.

Declared In `AdnDebugMgr.h`

Prototype `#define AdnDebugMessageIf (condition, messageP)`

Parameters → *condition*

A boolean value.

→ *messageP*

A pointer to a sequence of ASCII characters terminated by a null character (the string that is displayed in the debugger).

Returns Nothing.

See Also [AdnDebugMessage](#)

AdnDebugNativeRegister Macro

Purpose	This macro registers a PACE native object with the debugger.
Declared In	<code>AdnDebugMgr.h</code>
Prototype	<pre>#define AdnDebugNativeRegister (dbType, dbCreator, rsrcType, rsrcID)</pre>
Parameters	<p>→ <i>dbType</i> The application database type (for example, 'appl').</p> <p>→ <i>dbCreator</i> The application database's creator code.</p> <p>→ <i>rsrcType</i> The PACE native object's resource type (for example, 'ARMC').</p> <p>→ <i>rsrcID</i> The PACE native object's resource ID.</p>
Returns	Nothing.
Comments	The flag <code>kAdnEnableFullDebugging</code> must be set using AdnDebugEnableSet in order for <code>AdnDebugNativeRegister()</code> calls to be processed correctly.
See Also	AdnDebugEnableSet

AdnDebugNativeRegisterAddr Macro

Purpose	This macro registers a PACE native object with the debugger. This version can be used for code that is in a chunk separate from the application. Most PACE native objects can use the simpler form, AdnDebugNativeRegister .
Declared In	<code>AdnDebugMgr.h</code>
Prototype	<pre>#define AdnDebugNativeRegisterAddr (dbType, dbCreator, rsrcType, rsrcID, codeAddr)</pre>
Parameters	<p>→ <i>dbType</i> The application database type (for example, 'appl').</p> <p>→ <i>dbCreator</i> The application database's creator code.</p>

AdnDebug Manager

AdnDebugNativeUnregister

→ *rsrcType*

The PACE native's object resource type (for example, 'ARMC').

→ *rsrcID*

The PACE native's object resource ID.

→ *codeAddr*

The PACE native object's code base address.

Returns Nothing.

Comments The flag `kAdnEnableFullDebugging` must be set using [AdnDebugEnableSet](#) in order for `AdnDebugNativeRegisterAddr()` calls to be processed correctly.

AdnDebugNativeUnregister Macro

Purpose This macro unregisters a PACE native object with the debugger.

Declared In `AdnDebugMgr.h`

Prototype `#define AdnDebugNativeUnregister ()`

Parameters None.

Returns Nothing.

Comments You should call this macro prior to unlocking the PACE native object code resource so that the debugger breakpoints can be removed.

See Also [AdnDebugNativeRegister](#), [AdnDebugNativeRegisterAddr](#)

AdnDebugNativeUnregisterAddr Macro

Purpose This macro unregisters a PACE native object with the debugger.

Declared In `AdnDebugMgr.h`

Prototype `#define AdnDebugNativeUnregisterAddr (rsrcID)`

Parameters → *rsrcID*

A pointer to the `DbgPostUnloadParamsType` structure.

Returns Nothing.

Comments You should call this macro prior to unlocking the PACE native object code resource so that the debugger breakpoints can be removed.

See Also [AdnDebugNativeRegister](#), [AdnDebugNativeRegisterAddr](#)

AdnDebugUpdateLoadedModules Macro

Purpose Notify Palm OS Debugger of any recently loaded or unloaded native modules. This call is not used for debugging PACE native objects.

Declared In `AdnDebugMgr.h`

Prototype `#define AdnDebugUpdateLoadedModules ()`

Parameters None.

Returns Nothing.

AdnDebug Manager

AdnDebugUpdateLoadedModules

Index

Numerics

68K disassembler menu 68

A

append symbolics menu 86
ARM disassembler menu 68
arrange icons 90
ASCII integer menu 63

B

binary menu 63
book organization iii
boolean menu 63
breakpoints view 58
bytes per column menu 68
bytes per row menu 68

C

cascade menu 90
change filename menu 57
change path menu 57
char menu 64
clear all breakpoints menu 61
clear breakpoint menu 61
clear menu 74, 77
compiler requirements 3
conceptual overview 1
connect menu 85
control menu 88
copy all menu 58, 62, 64, 66, 67, 69, 70, 73, 74
copy menu 58, 62, 64, 66, 67, 69, 70, 73, 74, 81
cut menu 81

D

debug console 75
debug targets 2, 5
debugger preferences dialog 83
decimal (signed) menu 64
decimal (unsigned) menu 64
delete expression menu 73
disable all breakpoints menu 61
disable breakpoint menu 61

disassemble menu 52
disassembly menu 52
disconnect menu 86

E

echo input menu 74, 77
edit menu 81
enable all breakpoints menu 61
enable breakpoint menu 60
exit menu 80
exporting
 preferences 24
expressions console 70, 73

F

file menu 79
files view 55
find menu 82
find next menu 82
find previous menu 82
fixed 50/50 (signed) menu 64
fixed 50/50 (unsigned) menu 64
fixed 75/25 (signed) menu 64
fixed 75/25 (unsigned) menu 64
float menu 64
format changes are global menu 64
format elements as menu 66, 67
format menu 68
format types menu 66, 67

G

global variables view 66
goto PC menu 54

H

help menu 92
hex byte stream menu 64
hex menu 64
hex with decimal (signed) menu 64
hex with decimal (unsigned) menu 64
hexadecimal numbers
 base prefix 68
 uppercase 68

I

importing
 preferences 24

K

key bindings editor 82
key bindings menu 82
kill menu 89

L

load memory menu 86

M

memory view 67
menu reference 79
menus
 68K disassembler 68
 ARM disassembler 68
 ASCII integer 63
 binary 63
 boolean 63
 bytes per column 68
 bytes per row 68
 change filename 57
 change path 57
 char 64
 clear 74, 77
 clear all breakpoints 61
 clear breakpoint 61
 control 88
 control>kill 89
 control>restart 88
 control>run 88
 control>step 89
 control>step in 89
 control>step out 89
 control>stop 89
 copy 58, 62, 64, 66, 67, 69, 70, 73, 74
 copy all 58, 62, 64, 66, 67, 69, 70, 73, 74
 decimal (signed) 64
 decimal (unsigned) 64
 delete expression 73
 disable all breakpoints 61
 disable breakpoint 61
 disassemble 52

disassembly 52
echo input 74, 77
edit 81
edit>copy 81
edit>cut 81
edit>find 82
edit>find next 82
edit>find previous 82
edit>key bindings 82
edit>next possible breakpoint 82
edit>paste 82
edit>preferences 83
edit>previous possible breakpoint 82
edit>undo 81
enable all breakpoints 61
enable breakpoint 60
file 79
file>exit 80
file>open 80
file>save 80
fixed 50/50 (signed) 64
fixed 50/50 (unsigned) 64
fixed 75/25 (signed) 64
fixed 75/25 (unsigned) 64
float 64
format 68
format changes are global 64
format elements as 66, 67
format types 66, 67
goto PC 54
help 92
hex 64
hex byte stream 64
hex with decimal (signed) 64
hex with decimal (unsigned) 64
mixed for file 52
mixed for function 52
mixed for source line 52
new expression 73
number disassembler 69
open file in new window 57
paste 77
remove all added columns 69
return to current PC 52
revert to original path 58
run to cursor 54
select all 74, 77

- set breakpoint at address 53, 61
- set breakpoint at cursor 54
- set breakpoint at function 53, 61
- set disassembler 53
- set PC at cursor 54
- show all files 58
- show all globals 67
- show base prefix 68
- show file globals 66
- source 51
- string (C) 64
- string (Pascal) 64
- sync on eol 74, 77
- target 85
- target>append symbolics 86
- target>connect 85
- target>disconnect 86
- target>load memory 86
- target>remove symbolics 86
- target>save memory 87
- thumb disassembler 69
- uppercase numbers 68
- view 84
- view>status bar 85
- view>toolbar 85
- window 89
- window>arrange icons 90
- window>cascade 90
- window>tile 90
- mixed for file menu 52
- mixed for function menu 52
- mixed for source line menu 52

N

- new expression menu 73
- next possible breakpoint menu 82
- number disassembler menu 69

O

- open file in new window menu 57
- open menu 80
- operating systems 2

P

- Palm Debugger 2

- Palm OS Debugger
 - breakpoints view 58
 - comparison to Palm Debugger 2
 - debug console 75
 - debug targets 5
 - expressions console 70, 73
 - features 1
 - files view 55
 - global variables view 66
 - memory view 67
 - menus 79
 - overview 1
 - prerequisites 2
 - compiler 3
 - debug targets 2
 - operating systems 2
 - processes view 69
 - registers view 62
 - source view 48
 - stack trace view 70
 - stdio console 73
 - targets 5
 - variables view 64
 - windows 45
- paste menu 77, 82
- preferences
 - exporting 24
 - importing 24
 - resizing the preference fields 24
- preferences menu 83
- previous possible breakpoint menu 82
- processes view 69

R

- registers view 62
- remove all added columns menu 69
- remove symbolics menu 86
- resizing
 - preference fields 24
- restart menu 88
- return to current PC menu 52
- revert to original path menu 58
- run menu 88
- run to cursor menu 54

S

- save memory menu 87
- save menu 80
- select all menu 74, 77
- set breakpoint at address menu 53, 61
- set breakpoint at cursor menu 54
- set breakpoint at function menu 53, 61
- set disassembler menu 53
- set PC at cursor menu 54
- shortcut editor 82
- show all files menu 58
- show all globals menu 67
- show base prefix menu 68
- show file globals menu 66
- source menu 51
- source view 48
- stack trace view 70
- status bar menu 85
- stdio console 73
- step in menu 89
- step menu 89
- step out menu 89

- stop menu 89
- string (C) menu 64
- string (Pascal) menu 64
- sync on eol menu 74, 77

T

- target menu 85
- thumb disassembler menu 69
- tile menu 90
- toolbar menu 85

U

- undo menu 81
- uppercase numbers menu 68

V

- variables view 64
- view menu 84

W

- window menu 89