



Palm OS[®] Protein C/C++ Compiler Tools Guide

Palm OS[®] Developer Suite

Written by Eric Shepherd and Brian Maas

Technical assistance from Kevin MacDonell, Kenneth Albanowski, Flash Sheridan, Jeff Westerinen

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, 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.

Palm OS Protein C/C++ Compiler Tools Guide

Document Number 3123-002

November 15, 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

Table of Contents

About This Book	v
How This Book Is Organizedv
Palm OS Developer Suite Documentation	vi
Additional Resources	vii
1 Understanding Palm OS Application Development	1
Building a Palm OS Application1
Building a Palm OS Shared Library4
2 Introducing Palm OS Compiler Tools	7
Compiler Chain: pacc, paasm, palink8
Palm OS Librarian: palib9
Diagnostic Tool: elfdump	10
3 Using the Palm OS Compiler Chain	11
Palm OS Protein C/C++ Compiler	12
Compiler Command Line Interface	12
Compiler Options	13
Palm OS Assembler	21
Differences Between the Palm OS Assembler and the ARM	
Assembler	21
Assembler Command Line Interface	22
Assembler Options	22
Palm OS Linker	24
Linker Command Line Interface	24
Linker Options.	24
4 Using the Palm OS Librarian	29
Using the palib Command Line Tool	29
Creating a New Archive Library	29
Adding an ELF Object File to a Library	30
Deleting an ELF Object File from a Library.	30
Replacing an ELF Object File in a Library	30
Extracting an ELF Object Files from a Library	31
Displaying the Contents of a Library	31

palib Reference	32
Librarian Command Line Interface	32
Librarian Options	33
5 Using the Palm OS Shared Library Tool	35
Palm OS Shared Library Tool Concepts	36
Building Files for Device Targets	37
Building Files for Palm OS Simulator Targets.	37
Using pslib with Palm OS Developer Suite	38
Using the pslib Command Line Tool	38
Specifying Command Line Options.	39
6 Using the Palm OS Post Linker	43
Palm OS Post Linker Concepts	44
Using pelf2bin with Palm OS Developer Suite	45
Using the pelf2bin Command Line Tool	45
Specifying Command Line Options.	45
7 Shared Library Definition File Format Reference	47
Creating a Shared Library Definition File	48
Specifying Keywords	48
Sample Shared Library Definition Files	50
8 Using elfdump	53
Using the elfdump Command Line Tool	53
elfdump Reference	54
elfdump Command Line Interface	54
elfdump Options	55
Index	57

About This Book

This book describes the Palm OS C/C++ Protein Compiler tools:

- Palm OS C/C++ compiler, pacc
- Palm OS assembler, paasm
- Palm OS linker, palink
- Palm OS librarian, palib
- Diagnostic tool, elfdump

The audience for this book is application developers who want to write Palm OS applications using the C or C++ programming language for ARM-based handheld devices.

How This Book Is Organized

This book has the following organization:

- [Chapter 1, “Understanding Palm OS Application Development,”](#) on page 1 provides a general overview of the Palm OS application development process and explains how the Palm OS C/C++ Compiler tools can be used to build Palm OS applications.
- [Chapter 2, “Introducing Palm OS Compiler Tools,”](#) on page 7 provides an overview on how you can use the compiler tools to build code resources for Palm OS applications.
- [Chapter 3, “Using the Palm OS Compiler Chain,”](#) on page 11 describes how to use the command line version of the C/C++ compiler to build ELF object files from C and C++ source files.
- [Chapter 4, “Using the Palm OS Librarian,”](#) on page 29 describes how to build an library of ELF object files that you can use to manage your compiled code.
- [Chapter 5, “Using the Palm OS Shared Library Tool,”](#) on page 35 describes how to define the entry point and exports for Palm OS applications and shared libraries.
- [Chapter 7, “Shared Library Definition File Format Reference,”](#) on page 47 provides reference information on the shared library definition (SLD) file format.

About This Book

Palm OS Developer Suite Documentation

- [Chapter 6, “Using the Palm OS Post Linker,”](#) on page 43 describes how to use the Palm OS post linker as part of the build process.
- [Chapter 8, “Using elfdump,”](#) on page 53 describes how you can use the elfdump tool to inspect the contents of ELF object files.

Palm OS Developer Suite Documentation

The following tools books are part of the Palm OS Developer Suite package:

Document	Description
<i>Introduction to Palm OS Developer Suite</i>	Provides an overview of all of the Palm OS development tools: <ul style="list-style-type: none">• Compiler Tools• Resource Tools• Testing and Debugging Tools
<i>Palm OS Protein C/C++ Compiler Tools Guide</i>	Describes the tools associated with the Palm OS Protein C/C++ Compiler.
<i>Palm OS Protein C/C++ Compiler Language and Library Reference</i>	Provides reference information about the C and C++ languages and runtime libraries used with the Palm OS Protein C/C++ Compiler.
<i>Palm OS Debugger Guide</i>	Describes how to use Palm OS Debugger.
<i>Palm OS Resource Editor Guide</i>	Describes how to use Palm OS Resource Editor to create XRD files.

Document	Description
<i>Palm OS Resource Tools Guide</i>	Describes how to use the Palm OS resource tools: <ul style="list-style-type: none">• GenerateXRD - migration tool• Palm OS Resource Editor - XRD editor• PalmRC - building tool• PRCMerge - building tool• PRCCompare - comparison tool• hOverlay - localization tool• PRCSign and PRCCert - code-signing tools
<i>Palm OS Resource File Formats</i>	Describes the XML formats used for XML resource definition (XRD) files. XRD files are used to define Palm OS resources, and are the input files for the Palm OS resource tools.
<i>Palm OS Cobalt Simulator Guide</i>	Describes how to use Palm OS Cobalt Simulator.
<i>Palm OS Virtual Phone Guide</i>	Describes how to use Virtual Phone.

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>

About This Book

Additional Resources

- 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/>

Understanding Palm OS Application Development

This chapter gives you an overview of the application development process for Palm OS[®], describing how to use the compiler, linker, shared library tool, and post linker to develop applications.

NOTE: This overview is a simplification of the entire Palm OS application development process, with an emphasis on how the developer tools convert source files into an executable application. For a more complete description, see *Exploring Palm OS: Programming Basics*.

Building a Palm OS Application

When you write a Palm OS application, you generally need to define three things:

- The program logic. Most programs for Palm OS are written in C or C++. These source files are compiled into code resources.
- The user interface controls and data. Palm OS Protein application user interfaces are written in an XML format. XML Resource Definition (XRD) files are compiled into temporary resource (TRC) files.

Understanding Palm OS Application Development

Building a Palm OS Application

- Optionally define the entry points to the application. Palm OS Protein applications can have multiple entry points, though most will have a single entry point.

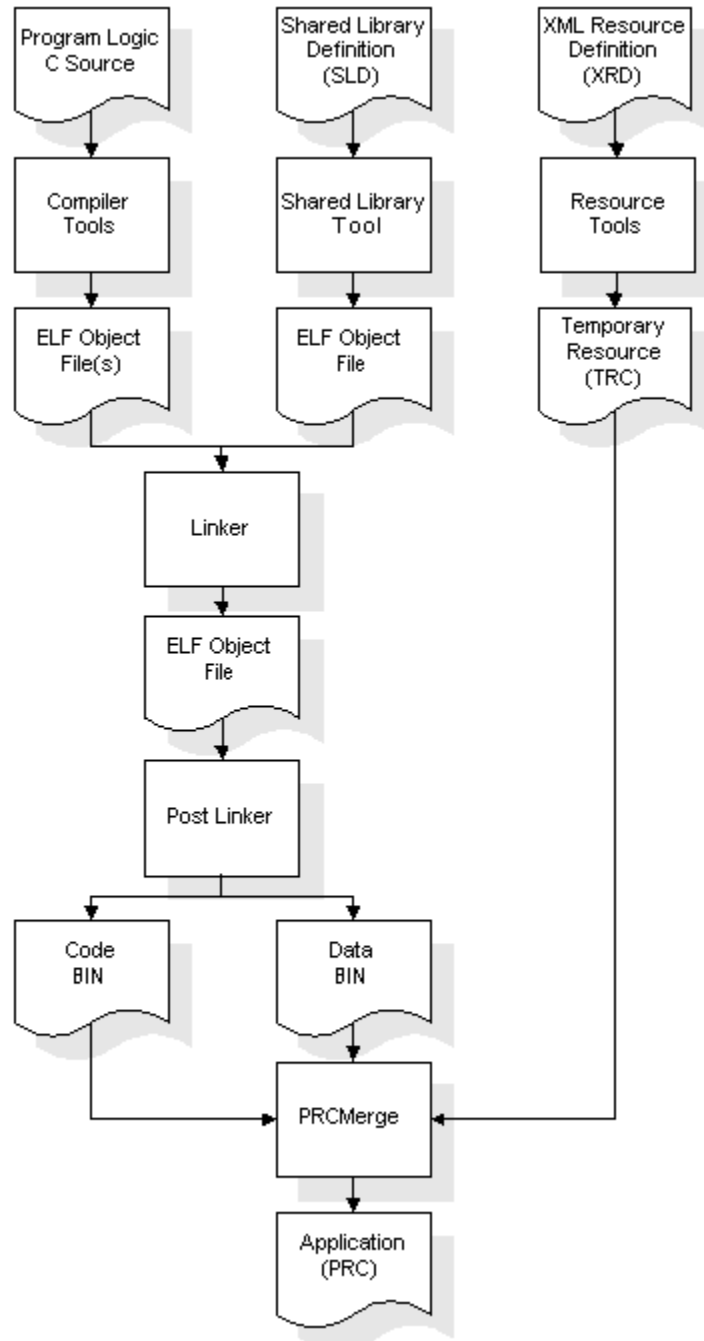
If your application has a single entry point, you can use the `PilotMain()` function as described in the book *Exploring Palm OS: Programming Basics*.

If your application has multiple entry points, you need to create a Shared Library Definition (SLD) file. For an application, your SLD file's first entry point is `_PalmUIAppStartup`; this entry point will call your application's `PilotMain()` function. Your other entry points can have arbitrary C prototypes. For more information on SLD files, see [Chapter 7, "Shared Library Definition File Format Reference,"](#) on page 47.

NOTE: Entry points can only be C functions; C++ methods cannot be used as entry points.

[Figure 1.1](#) on page 3 provides an overview of the build process.

Figure 1.1 Palm OS Application Development Overview



Understanding Palm OS Application Development

Building a Palm OS Shared Library

As shown in [Figure 1.1](#), these developer tools are used in the build process:

- The compiler tools compile the C source files into ELF object files. The compiler tools are described in this book.

For more information about the compiler tools, see [Chapter 2, “Introducing Palm OS Compiler Tools,”](#) on page 7.

- For applications, the shared library tool compiles the shared library definition (SLD) file into a single ELF object file. The shared library tool, `pslib`, is described in this book.

For more information about `pslib`, see [Chapter 5, “Using the Palm OS Shared Library Tool,”](#) on page 35.

- The resource tools, specifically `PalmRc`, compile the XML Resource Definition (XRD) file into a temporary resource (TRC) file. For more information about the resource tools, see *Palm OS Resource Tools Guide*.

- The linker combines ELF object files into a single ELF object file. For more information about the linker, see [“Palm OS Linker”](#) on page 24.

- The post linker converts the ELF object file into binary resource files that can be merged into a Palm OS application. The post linker, `pe1f2bin`, is described in this book.

For more information about `pe1f2bin`, see [Chapter 6, “Using the Palm OS Post Linker,”](#) on page 43.

- One of the resource tools, `PRCMerge`, combines the code resource, data resource, and temporary resource files into the final Palm OS application (PRC) file. For more information about `PRCMerge`, see *Palm OS Resource Tools Guide*.

Building a Palm OS Shared Library

The process for building a Palm OS shared library is similar to the process for building a Palm OS application. However, for shared libraries, the shared library definition (SLD) file defines a unique entry point and generally defines multiple exports from the library.

When `pslib` compiles the SLD file for a library, it produces two object files:

- An ELF object file containing the startup code for each library function. This object file is linked together with the library's object files that you get from compiling the C or C++ source code.
- An ELF object file containing the stub code for each library function. This object file is linked with the program that calls the library's function.

Understanding Palm OS Application Development

Building a Palm OS Shared Library

Introducing Palm OS Compiler Tools

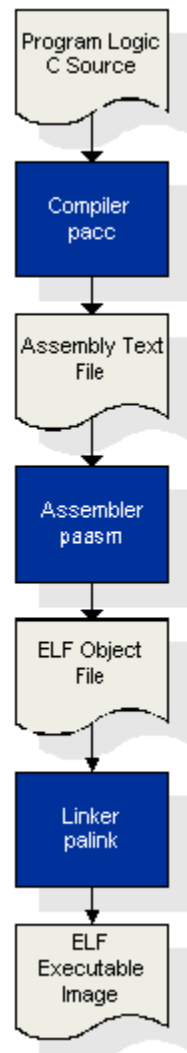
This chapter describes the compiler tools that you can use to build code resources for Palm OS applications:

- “[Compiler Chain: pacc, paasm, palink](#)” on page 8 describes the basic tools in the compiler chain.
- “[Palm OS Librarian: palib](#)” on page 9 provides an overview of the Palm OS librarian tool.
- “[Diagnostic Tool: elfdump](#)” on page 10 introduces how you can inspect ELF object file contents with the `elfdump` tool.

Compiler Chain: *pacc*, *paasm*, *palink*

As is common with command line compilers, the Palm OS Protein C/C++ Compiler, *pacc*, acts as a driver. *pacc* invokes all of the commands necessary to produce linked files from source code.

Figure 2.1 Compiler Chain Overview



- *pacc* compiles the source files into assembly language source files.

- pacc calls the assembler, paasm, to produce ELF object files from the assembly language source files.
- pacc calls the linker, palink, to generate the ELF executable image from the ELF object files.

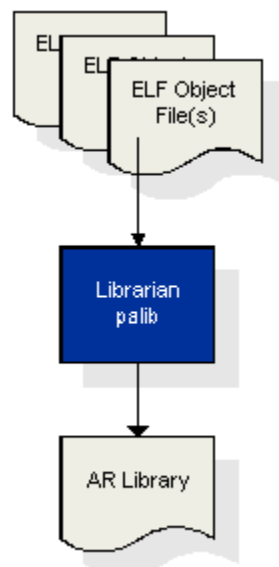
For more information about using the compiler chain, see [Chapter 3, “Using the Palm OS Compiler Chain,”](#) on page 11.

IMPORTANT: It's important to note that paac and palink are only used when compiling for ARM processors. When building to run in the Palm OS Simulator, the gcc compiler is used instead, to build the necessary x86 executable code.

Palm OS Librarian: palib

The Palm OS librarian tool, palib, lets you create and manage a collection of ELF object files. palib creates library files that conform to the Unix 'ar' archive file format.

Figure 2.2 Librarian Overview



With palib, you can:

Introducing Palm OS Compiler Tools

Diagnostic Tool: elfdump

- Create a new archive library.
- Add ELF object files to the library.
- Delete ELF object files from the library.
- Replace ELF object files in a library.
- Extract ELF object files from a library.

For more information about `palib`, see [Chapter 4, “Using the Palm OS Librarian,”](#) on page 29.

Diagnostic Tool: elfdump

`elfdump` lets you extract the contents of an ELF object file into a text file. With `elfdump`, you can:

- Disassemble executable bytecode sections.
- Disassemble data sections as code.
- Disassemble for a given instruction set architecture.
- Show only segment and section summaries.
- Show specific sections, such as code, data, debug information or symbols.

For more information about `elfdump`, see [Chapter 8, “Using elfdump,”](#) on page 53.

NOTE: This tool is included in the compiler suite because you may find it useful, but it is an unsupported tool. It is only available from the command line.

Using the Palm OS Compiler Chain

The Palm OS compiler chain consists of the following tools:

- [“Palm OS Protein C/C++ Compiler”](#) on page 12
- [“Palm OS Assembler”](#) on page 21
- [“Palm OS Linker”](#) on page 24

Palm OS Protein C/C++ Compiler

The Palm OS Protein C/C++ Compiler is a full-featured, standards-based, optimizing C/C++ compiler.

- The Palm OS compiler supports the C++ language standard ANSI/ISO 14882:1998(E).
- The Palm OS compiler supports the C language standard ANSI/ISO/IEC 9899:1999, commonly known as “C99.”

The compiler, `pacc`, takes one or more C/C++ language text files as input, and produces a corresponding number of assembly language source files as output. Optionally, `pacc` assembles the assembly language files into object code by calling Palm OS Assembler, and links the object code files into an ARM executable file by calling Palm OS Linker.

NOTE: The Palm OS Protein C/C++ Compiler supports both common C/C++ keyword extensions (see “[Keywords](#)” on page 14 of the book *Palm OS Protein C/C++ Compiler Language and Library Reference*) as well as several predefined macros specific to the `pacc` compiler (see “[Preprocessor Directives](#)” on page 20 of the *Palm OS Protein C/C++ Compiler Language and Library Reference*).

Compiler Command Line Interface

The general format of the `pacc` command line interface is this:

```
pacc [options] source_file [source_files]
```

`options`

Compiler options, as described in the section “[Compiler Options](#)” on page 13.

`source_file [source_files]`

`pacc` supports the following types of input files:

`.c`

C source program.

.cc, .cxx, .cp, .c++, .cpp

A C++ source program. The `.c++` extension is not recognized by the Palm OS Developer Suite, even though the compiler supports it.

.s

An assembly source program, as input to the Palm OS assembler.

.o

A relocateable object file, as input to the Palm OS linker.

.l, .a, .lib

A library object file, as input to the Palm OS linker.

NOTE: These file extensions are accepted regardless of case.

Compiler Options

`pacc` has options which control its behavior, as is standard for all compilers. The following compiler options are supported.

-c

The compiler stops the compilation before invoking the linker, leaving the object (`.o`) files in the current directory. Any source files are compiled and/or assembled into an object file.

Use the compiler option `-o` to specify the output object file name.

-C

`pacc` retains comments in the C preprocessor output, when used with `-E` or `-P` option.

-D *string*

`pacc` defines names as specified by *string*. This option applies only to source files passed through the C preprocessor.

Note: Whitespace is optional between `-D` and *string*.

Using the Palm OS Compiler Chain

Palm OS Protein C/C++ Compiler

string

Can be of the form `name=def` or `name`.

In the first case, `name` is defined with value `def` exactly as if a corresponding `#define` statement is the first line of the program.

In the second case, `name` is defined with the value 1.

The `-D` option has a lower precedence than the `-U` option, which is described below.

`-E`

`pacc` stops after preprocessing source.

For this option, `pacc` preprocesses any source files, writing the output either to `stdout`, or to the file specified with by the compiler option `-o`, which is described below.

The preprocessor removes comment lines by default. To retain comment lines, use the compiler option `-C`, which is described above.

`-ex`

This option enables `pacc`'s exception handling support.

`-g`

`pacc` includes symbolic debugging information in the assembly files, and sets the default optimization level to `-O1`.

See also the compiler option `-g0`, described below.

`-g0`

Note: "0" is the number zero.

This compiler option is similar to the option `-g`, but `pacc` inlines functions declared with the `inline` specifier.

This option usually improve run-time speed and reduces code size, but may make it more difficult to debug inline functions.

`-I dir`

This option changes the search path used to find files named in the C `#include` statements.

NOTE: Whitespace is optional between `-I` and `dir`.

The search order for `#include` statements is defined as follows:

1. For filenames that are absolute pathnames, `pacc` uses only the filename as specified.
2. For filenames that are not absolute pathnames and that are enclosed in quotation marks (" "), `pacc` searches relative to the following directories, in the listed order:
 - a. The directory containing the source file that contains the `#include` statement.
 - b. The directories listed in any `-I` compiler options, in the order the options occur on the command line.
 - c. The directories where the `pacc` standard headers have been installed.
3. For file names that are not absolute pathnames and that are enclosed in angle brackets (< >), `pacc` searches relative to the following directories, in the listed order:
 - a. The directories listed in any `-I` compiler options, in the order the options occur on the command line.
 - b. The directories where the `pacc` standard headers have been installed.

`-Ldir`

This option specifies a library path, which is passed to the linker via the `palink -libpath` option. `palink` uses the directory specified by `dir` to look for libraries that cannot otherwise be found.

If you specify this option without a directory, then `palink` will not search the default directories.

NOTE: Do not use any whitespace between `-L` and `dir` when you specify this option.

`-logo`

`pacc` displays the logo banner, consisting of the version and copyright notice, on each run. This is the default setting.

To turn this feature off, use the compiler option `-nologo`.

Using the Palm OS Compiler Chain

Palm OS Protein C/C++ Compiler

-noex

This option disables `pacc`'s exception handling support. This is the default setting.

To enable exception handling support, use the compiler option `-ex`.

-nologo

`pacc` does not display the logo banner.

-nostackwarn

Disables stack size warnings. This is the same as `-stackwarn=0`.

-o *outfile*

Use this option to set the name of the output file to something other than what the default rules would have generated.

Certain restrictions on the suffix of *outfile* are enforced if compilation is stopped before calling the linker, `palink`. This restriction prevents accidental overwriting of the source file, for instance.

NOTE: You must have whitespace between `-o` and *outfile* when you specify this option.

-O

Note: "O" is the capital letter "o".

`pacc` sets the optimization level to the generally useful level of global optimization. This option is an abbreviation for the compiler option `-O3`.

-On

Note: "O" is the capital letter "o".

`pacc` sets the optimization to the value specified by *n*, where *n* is a number between zero (0) and five (5).

0 (zero)

No significant optimization; the compiler may perform very basic optimizations but generally does not.

1

Local (basic-block scope) optimization of blocks, only.

- 2 The same as option `-O1`, plus intraprocedural global optimization, scheduling, and variables may reside in registers.
- 3 The same as option `-O2`, plus more extensive global optimizations.
- 4 The same as `-O3`, plus interprocedural global optimization and inlining.
- 5 The same as `-O4`, plus more extensive inlining and global optimizations.

Interprocedural optimization only applies to multiple C files compiled to object files within a single invocation of `pacc`.

You should be careful when handling object (`.o`) files produced by the options `-O4` and `-O5`. In these modes, when multiple files are passed to the compiler, interprocedural optimization occurs across files, so the resultant object files are dependent on each other for correct execution. If you make a change in one of these source files, you must recompile all of the related files.

The default level of optimization is `-O1`.

NOTE: You must not have any whitespace between `-O` and *n* when you specify this option.

`-P`

`pacc` preprocesses all C/C++ source files, with the preprocessing result for each file written to a file name that has the file extension `.i` substituted for the file name suffix of the source file.

The preprocessor removes comment lines by default. To retain comment lines, use the compiler option `-C`, which is described above.

`--preinclude=filename`

Each `--preinclude` argument supplies a filename that will be implicitly included in each compiled source file, as if there

Using the Palm OS Compiler Chain

Palm OS Protein C/C++ Compiler

were a corresponding `#include` directive at the beginning of the source file. There must not be a space between `--preinclude=` and the filename.

`-S`

`pacc` stops after producing assembly from C/C++ source. Any source files are compiled as far as an assembly language (`.s`) file. Use `-o` to specify the output assembly language filename. `pacc` stops the compilation before invoking the assembler and leaves all of the assembly source files produced by the compilation in the current directory.

`-stackwarn`

Sets the stack warning size to 8,192 bytes. This is the same as `-stackwarn=8192`.

`-stackwarn=n`

Sets the stack warning size to *n* bytes, where *n* is an integer. If any function allocates more stack than this value, a warning will be emitted describing how much stack the function would use. If *n* is 0, stack warning is disabled. The default value is 8,192, which results in a warning for functions using more than 8K of stack space.

`-strict`

`pacc` is more strict about ANSI rules when compiling C/C++ source code, and emits error messages for behavior that is unsupported by the ANSI standard.

Use the compiler option `-Wstrict` if you want `pacc` to treat these errors as warnings.

`-U name`

`pacc` undefines the name specified by *name*. This option applies only to source files passed through the C/C++ preprocessor.

The `-U` option overrides a `-D` option for the same name regardless of the order of the options on the command line. Any initial definition of *name* is removed.

NOTE: Whitespace is optional between `-U` and *name*.

`-V`

`pacc` writes its version numbers to `stderr`, and exits without performing any further actions.

-v

`pacc` uses verbose output, showing all commands used for compilation, assembly, and linking.

-vv

`pacc` uses verbose output, showing all commands used for compilation, assembly, and linking, but does not execute the commands.

-w

Use `-w` to suppress all warning messages from compiler and preprocessor. This option suppresses warnings from preprocessors, but not from the assembler or linker.

-wall

Use `-wall` to enable all warning messages from compiler and preprocessor. This is the default setting.

-wen

This option makes the message number, specified by *n*, into an error message.

NOTE: You must not have whitespace between `-we` and *n* when you specify this option.

-wdn

This option suppresses the warning or error number specified by *n*, if the message is suppressible. (Some errors are not suppressible.)

NOTE: You must not have whitespace between `-wd` and *n* when you specify this option.

-won

This option prevents the remark, warning, or suppressible error number, specified by *n*, from being emitted more than once, within a single source file.

NOTE: You must not have whitespace between `-wo` and *n* when you specify this option.

-Wn

This option suppresses messages, based on the value of *n*:

Using the Palm OS Compiler Chain

Palm OS Protein C/C++ Compiler

- 0
Suppresses all remarks, warnings, and suppressible errors
- 2
Suppresses only remarks
- 4
Suppresses nothing. All remarks, warnings, and errors are reported.

The default is 2. The option `-W0` is the same as the option `-w`. (The option `-W1` is treated the same as the option `-W2`, and the option `-W3` is treated as the option `-W4`.)

`-Werror`

`pacc` treats all compiler warnings as errors, so they prevent the compilation from succeeding. This option does not affect errors from the Palm OS assembler or Palm OS linker.

`-Wstrict`

`pacc` is less strict about compiling C/C++ source code with the ANSI rules, and issues warnings for behavior unsupported by the standard.

For example, the ANSI standard requires a semicolon to delineate items in a `struct` definition. In the code example below, the missing semicolon after `uint32_t item` is an error when the `-strict` option is used.

```
typedef struct {  
    uint32_t item  
} MyType;
```

However, with the `-Wstrict` option specified, this coding error is treated as a warning.

Palm OS Assembler

The Palm OS assembler, `paasm`, processes the assembly language text files produced by `pacc`, and produces binary object files conforming to the ARM-ELF standard (SWS ESPC 0003 B-01).

`paasm` recognizes and assembles the entire ARM 4T instruction set with the following exceptions:

- THUMB instructions

Palm OS Protein C/C++ Compiler is not a Thumb compiler, but Thumb is specified as part of the 4T architecture.

- MRS/MSR

There is no support for the instructions that read and write the status register.

- LDRT/STRT

These are only useful for privileged exception handlers.

- LDM(2), LDM(3) and STM(2)

These are unpredictable in User or System modes.

As a developer, you do not generally use this program directly. Rather, `pacc` compiles source files and calls `paasm` for you.

NOTE: This assembler is intended for assembling output of Palm OS Protein C/C++ Compiler, `pacc`. This is not a general purpose assembler; it does not support assembling manually-created assembly language programs.

However, in certain debugging situations, you may be interested in inspecting the assembly files before they are converted into ARM-ELF binary object files.

Differences Between the Palm OS Assembler and the ARM Assembler

There are several differences between the Palm OS assembler and that provided by ARM in its development suite:

- The Palm OS assembler requires that all opcodes be in lower case.

Using the Palm OS Compiler Chain

Palm OS Assembler

- Opcodes do not need to be indented.
- Labels must be terminated with a colon.
- Labels are only available for use with directives and cannot be used for references in opcode parameters.
- The directives are completely different.
- An ARM assembly file must begin and end with `area` and `end` directives; the Palm OS assembler rejects those directives.
- “&” to indicate a hexadecimal literal is not supported by `paasm`. Neither is “2_” to indicate a binary literal, nor “n_” to indicate other bases.
- Branches to `<label> + <number>` are not supported by `paasm`.
- References of the form “`mov r2, #label`”, where “`label`” is a label, are not supported.
- Some opcode/register combination instructions are accepted by the ARM assembler (with unpredictable results) but are rejected by the Palm OS assembler.

NOTE: The Palm OS Assembler is not intended for use other than by the C/C++ Compiler. PalmSource™ does not recommend using it to directly write assembly language code.

Assembler Command Line Interface

The general format of the Palm OS assembler command line interface is this:

```
paasm [options] asmfile.s
```

options

Assembler options, as described in the section “[Assembler Options](#)” on page 22.

Assembler Options

```
-o outputFileName
```

Specifies the output ARM-ELF file name.

-V

paasm writes the its version numbers to stderr, and exits without performing any further actions.

Palm OS Linker

The Palm OS linker combines linkable ARM-ELF object files into a single ARM executable file. As a developer, you do not generally use this program directly. Rather, the `pacc` calls `palink` for you.

However, in certain situations, you may want to run the linker independent from the compiler. For example, you may be interested in changing linker options for debugging reasons without wanting to recompile source into object files.

Linker Command Line Interface

The general format of the Palm OS linker command line interface is this:

```
palink [options] inputFiles
```

`options`

Linker options, as described in the section “[Linker Options](#)” on page 24.

`inputFiles`

Space-separated list of object files or libraries. Input files are put into output in the order given.

Linker Options

`-help`

`palink` prints a summary of help.

`-d` | `-debug`

`palink` includes debug information (debug input sections and the symbol and string tables) in the output file.

This is the default setting. To turn off this option, use the option `-nodebug`.

`-entry location`

`palink` uses the given numeric value or a symbol specified by `location` as the unique entry point of the output file.

`-errors file`

Use this option to tell `palink` to redirect error output to the specified `file` instead of using `stderr`.

-first *sectionid*

Use this option to tell `palink` that the section specified by *sectionid* is to be placed first in the output file.

-info *topics*

`palink` displays information on specific items, defined by *topics*:

sizes

`palink` gives a list and the totals of the code and data sizes (for read-only data, read-write data, zero-initialized data, and debug data) for each input object and library member in the ELF object file. Using this option is equivalent to using this option:

-info sizes,totals.

totals

`palink` gives the totals of the code and data sizes (for read-only data, read-write data, zero-initialized data, and debug data) for input objects and libraries.

unused

`palink` lists all unused sections that were eliminated when the output file was created.

These *topics* can be specified alone or can be used together, separated by commas but with no spaces:

-info sizes,totals,unused

-libpath *pathlist*

This option instructs `palink` where to search for library files when an unqualified library file does not exist in the current working directory.

pathlist

Specifies a list of directories. *pathlist* must contain at least one directory. *pathlist* is a comma-delimited list of directories. (The delimiter can only be a single comma with no intervening whitespace.)

You can specify this option multiple times; the resulting *pathlist* is the set of all directories you have specified.

Linker input files that are specified with path qualifiers are only searched in the resulting

Using the Palm OS Compiler Chain

Palm OS Linker

directories. Linker input files with no path qualification are first searched for in the current working directory then in each of the directories in the resulting *pathlist*, in sequential order.

-list *file*

Use this option to tell `palink` to redirect standard output to the specified *file*.

-locals

`palink` adds local symbols to the output symbol table.

This is the default setting. To turn off this option, use the option `-nolocal`s.

-mangled

`palink` uses object file values for the C++ symbols in error messages and in the text output created by the `-info`, `-map`, `-symbols`, and `-xref` options. The symbol table itself is not altered.

This option overrides the default option `-unmangled`.

-map

`palink` outputs an object file map.

-nodebug

`palink` does not include debug information in the output file.

This option overrides the default option `-debug`.

-nolocals

`palink` does not add local symbols to the output symbol table.

This option overrides the default option `-local`s.

-o *filename* | -output *filename*

`palink` sets the name of the output file to the name specified by *filename*.

The default output filename from `palink` is `elf.o`.

-symbols

`palink` outputs symbols that are used in the link step.

-unmangled

`palink` uses source code equivalents for the C++ symbols in error messages and in the text output created by the `-info`,

`-map`, `-symbols`, and `-xref` options. The symbol table itself is not altered.

This option is the default. To turn off this option, use the option `-mangled`.

`-v`

`palink` writes the its version numbers to `stderr`, and exits without performing any further actions.

`-via file`

Use this option to tell `palink` to read more options from the specified *file*.

`-xref`

Use this option to tell `palink` to create an intersectional cross-reference table.

Using the Palm OS Compiler Chain

Palm OS Linker

Using the Palm OS Librarian

The Palm OS librarian, `palib`, is a tool that you use to create and manage a collection of ELF object files. `palib` creates library files that conform to the Unix 'ar' archive file format.

- [“Using the palib Command Line Tool”](#) on page 29
- [“palib Reference”](#) on page 32

Using the palib Command Line Tool

With `palib`, you can do all of the following tasks:

- [Creating a New Archive Library](#)
- [Adding an ELF Object File to a Library](#)
- [Deleting an ELF Object File from a Library](#)
- [Replacing an ELF Object File in a Library](#)
- [Extracting an ELF Object Files from a Library](#)
- [Displaying the Contents of a Library](#)

Creating a New Archive Library

To create an archive library, you specify the option `-create`:

```
palib -create myLib.l
```

This command creates an empty library file with the name `myLib.l`.

As an alternative, you can create a library using the option `-add`:

```
palib -add myLib.l TestMain.o
```

This command creates `myLib.l` if it doesn't exist, and then adds `TestMain.o` to `myLib.l`.

Adding an ELF Object File to a Library

To add an ELF object file to an archive library, you specify the option `-add`:

```
palib -add myLib.1 TestsCode.o
```

This command adds `TestsCode.o` to `myLib.1` if it already exists, or creates `myLib.1` and adds `TestsCode.o` if the library file doesn't already exist.

NOTE: If the ELF object file is already a member of the library, then `palib` displays an error message and the file is not added to the library.

You can specify multiple ELF object files in one add request.

Deleting an ELF Object File from a Library

To remove an ELF object file from an archive library, you specify the option `-delete`:

```
palib -delete myLib.1 TestsCode.o
```

This command deletes `TestsCode.o` from `myLib.1`.

You can specify multiple ELF object files in one delete request.

Replacing an ELF Object File in a Library

To replace an ELF object file in an archive library, you specify the option `-replace`:

```
palib -replace myLib.1 TestsPlug.o
```

If the ELF object file `TestsPlug.o` is in the library `myLib.1`, this command replaces the `TestsPlug.o` file. If `TestsPlug.o` is not already in the library file, the command simply adds `TestsPlug.o` to `myLib.1`.

You can specify multiple ELF object files in one replace request.

Extracting an ELF Object Files from a Library

To extract an ELF object file from an archive library, you specify the option `-extract`:

```
palib -extract myLib.l TestsCode.o
```

If the ELF object file `TestsCode.o` is in the library `myLib.l`, this command extracts the `TestsCode.o` file to the local directory.

You can specify multiple ELF object files in one extract request.

WARNING! If the local directory already has a file by the same name as the one you are extracting, `palib` overwrites the existing file with the one extracted from the library file.

Displaying the Contents of a Library

To display a list of object files in a library, you specify the option `-toc`:

```
palib -toc TestLib.L
```

The output shows the list of object files in the order that you added them to the library.

Listing 4.1 Sample output from the option `-toc`

```
TestsLib_Startup.o
TestsPlug.o
TestsRendering.o
TestsCode.o
Tests.o
TestsLibMain.o
```

To display a list of symbols in the library, you specify the option `-symtab`:

```
palib -symtab TestLib.L
```

The output shows the list of symbols in the order in which they appear in the ELF object files.

Using the Palm OS Librarian

palib Reference

Listing 4.2 Sample output from the option `-symtab`

<code>__user_libspace</code>	from <code>TestsLib_Startup.o</code>	at offset 1474
<code>\$Sub\$\$TestSetFormId</code>	from <code>TestsLib_Startup.o</code>	at offset 1474
<code>\$Sub\$\$TestSetFormPtr</code>	from <code>TestsLib_Startup.o</code>	at offset 1474
<code>\$Sub\$\$TestSetGadgets</code>	from <code>TestsLib_Startup.o</code>	at offset 1474
<code>RenderDefineRoundRect</code>	from <code>TestsRendering.o</code>	at offset 27e9e
<code>RenderRawBitmapLabel</code>	from <code>TestsRendering.o</code>	at offset 27e9e
<code>RenderGetTextHeight</code>	from <code>TestsRendering.o</code>	at offset 27e9e
<code>PrvTestGadgetTabsBodyCallBack</code>	from <code>TestsCode.o</code>	at offset 45172
<code>PrvTestUpdateScrollFlag</code>	from <code>TestsCode.o</code>	at offset 45172
<code>TestSetFlags</code>	from <code>Tests.o</code>	at offset 651e6
<code>TestGetTextColors</code>	from <code>Tests.o</code>	at offset 651e6
<code>TestSetEnableUpdate</code>	from <code>Tests.o</code>	at offset 651e6
<code>TestGetTabGraphics</code>	from <code>Tests.o</code>	at offset 651e6
<code>TestsLibMain</code>	from <code>TestsLibMain.o</code>	at offset 8c51a

To display a list of entry points defined in the library, you specify the option `-entries`:

```
palib -entries TestLib.L
```

The output shows the list of entries in the order in which they appear in the ELF object files.

Listing 4.3 Sample output from the option `-entries`

```
ENTRY at offset 0 in section startup_code_header_area of TestsLib_Startup.o
ENTRY at offset 0 in section startup_code_header_area of SampleLib_Startup.o
```

palib Reference

This section provides reference information for the `palib` tool.

- [Librarian Command Line Interface](#)
- [Librarian Options](#)

Librarian Command Line Interface

The general format of the `palib` command line interface is this:

```
palib [options] libraryName [elfFileList]
```


options

palib options, as described in the section “[Librarian Options](#).”

libraryName

The name of the library (.L) file. If the library file exists, then palib will use the library specified; if the library file does not exist, then palib will create the file.

elfFileList

A list of ELF object files.

Librarian Options

-add | -a

palib adds the ELF object files specified by *elfFileList* to the library.

-create | -c

palib creates a new library, overwriting any existing library with the same name.

-delete | -d

palib deletes the files specified by *elfFileList* from the library.

-entries | -e

palib displays a list of entry points defined in a library.

-extract | -x

palib extracts the files specified by *elfFileList* from the library.

-help | -h

palib prints a summary of help.

-replace | -r

palib replaces the files specified by *elfFileList* in a library. If a file does not already exist, it will simply be added.

-syntab | -s

palib displays a table of all symbols and where they reside in the library.

-toc | -t

palib displays the table of contents of the library.

Using the Palm OS Librarian

palib Reference

-V

`palib` writes the its version numbers to `stderr`, and exits without performing any further actions.

-via *filename*

`palib` reads the file *filename* for more options.

Using the Palm OS Shared Library Tool

This chapter describes how you can use `pslib`, the Palm OS shared library tool, to define the entry point and exports for Palm OS applications and shared libraries.

- [“Palm OS Shared Library Tool Concepts”](#) on page 36
- [“Using pslib with Palm OS Developer Suite”](#) on page 38
- [“Using the pslib Command Line Tool”](#) on page 38

Palm OS Shared Library Tool Concepts

The Palm OS shared library tool `pslib` is essential for building Palm OS applications and shared libraries. [Chapter 1](#), “[Understanding Palm OS Application Development](#),” on page 1 provides an overview of the entire process for building Palm OS applications and shared libraries, and describes how `pslib` fits in the overall process.

This is the process for using `pslib`:

1. First, create a shared library definition (SLD) file. (See [Chapter 7](#), “[Shared Library Definition File Format Reference](#),” on page 47 for information on creating SLD files.)
2. Then use `pslib` to convert your SLD file into object files targeted for execution either on Palm OS devices or on Palm OS Simulator.

For more information about Palm OS device targets, see the section “[Building Files for Device Targets](#)” on page 37.

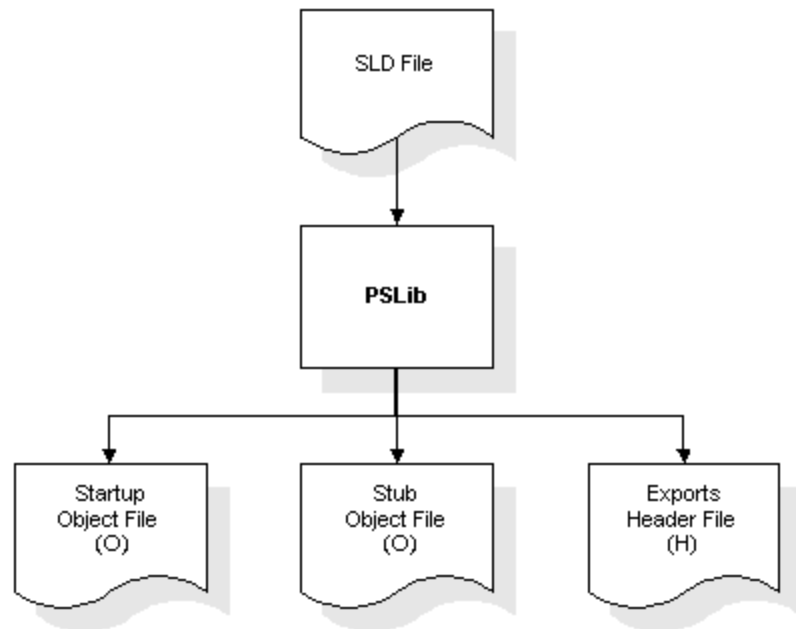
For more information about Palm OS Simulator targets, see the section “[Building Files for Palm OS Simulator Targets](#)” on page 37.

3. Link the startup object file created by `pslib` with your compiled code object files to produce your application or shared library.
4. Link the stub object file created by `pslib` with an application that calls a function exported by your shared library.

Building Files for Device Targets

[Figure 5.1](#) on page 37 shows the files that `pslib` produces for ARM-based device targets. When you build code to run on ARM-based devices, you need to link the Palm OS startup object file with the code you compile with the Palm OS compiler.

Figure 5.1 `pslib` Overview for Device Targets



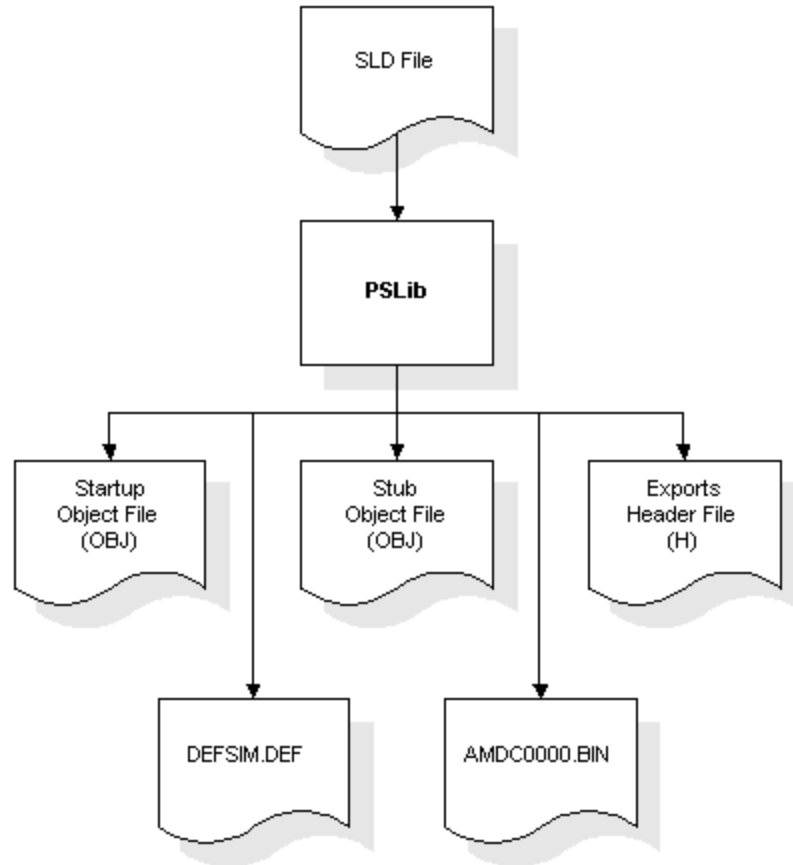
Building Files for Palm OS Simulator Targets

[Figure 5.2](#) on page 38 shows the files that `pslib` produces for Palm OS Simulator targets. When you build code to run on Palm OS Simulator, you need to link the startup object file with the code you compile with the `gcc` compiler for x86.

Using the Palm OS Shared Library Tool

Using pslib with Palm OS Developer Suite

Figure 5.2 pslib Overview for Palm OS Simulator Targets



Using pslib with Palm OS Developer Suite

`pslib` is fully integrated with Palm OS Developer Suite. When you build your application or shared library with Palm OS Developer Suite, `pslib` is called as part of the build process. You do not need to invoke `pslib` directly.

Using the pslib Command Line Tool

`pslib` is used to compile a shared library definition file (SLD) into binary resource files that can be linked into a Palm OS shared library or Palm OS application.

The command line syntax for *pslib* is:

```
pslib -inDef filename [options]
```

-inDef [*none* | *filename*]

The **-inDef** parameter is required, with either a filename or the value *none*.

none

When the value *none* is specified, no shared library definition file is required. All information is taken from the remaining command line options.

filename

The filename of the input shared library definition file. The input SLD file must conform to the format described in [Chapter 7, “Shared Library Definition File Format Reference,”](#) on page 47.

options

Additional command line options as described in the following section [“Specifying Command Line Options.”](#)

Specifying Command Line Options

-ARMarch [*4T* | *5T* | *5TE* | *0*]

This parameter specifies the minimum required ARM architecture to load this library. (This parameter does not apply to targets built for Palm OS Simulator.)

If you specify this optional parameter, it overrides the **ARMARCH** keyword in the SLD file.

pslib issues a warning message when this command line option and the SLD file value are different.

-creator *four_character_code*

four_character_code specifies a 4-byte resource type.

If you specify this optional parameter, it overrides the creator specification in the SLD file. *pslib* issues a warning message when this command line option and the SLD file value are different.

Using the Palm OS Shared Library Tool

Using the *pslib* Command Line Tool

-entry *entryName*

entryName specifies the name of the primary entry point.

If you specify this optional parameter, it overrides the **ENTRY** keyword in the SLD file. *pslib* issues a warning message when this command line option and the SLD file value are different.

-execName *executableName*

executableName overrides the name of the SLD file as the default base of the executable filename (the filename used for locating DLL with Palm OS Simulator.)

This parameter is optional unless the **-inDef** parameter specified none.

-help | **-h**

pslib displays help information and ignores any other options.

-OSversion *versionnumber*

versionnumber specifies the minimum required version of Palm OS to load this library. If you don't specify this option, Palm OS Cobalt 6.0 is assumed.

The version number is in the format major.minor.fix, then the stage ("d", "a", "b", or "r"), and the build number. For example: "6.0.1b34", "3", or "6.1r".

If you specify this optional parameter, it overrides the **OSVERSION** keyword in the SLD file.

pslib issues a warning message when this command line option and the SLD file value are different.

-outEntryNums *filename*

filename specifies the output C/C++ header file with enumerations (**enum**) and defines (**#define**) for each module entry point.

-outErrors *filename*

filename specifies the name of a file to which you want *pslib* to write error messages.

-outObjStartup *filename*

filename specifies the output startup object filename for a Palm OS device target build.

- `-outObjStubs filename`
filename specifies the output stubs object filename for a Palm OS device target build.
- `-outSimDefs filename`
filename specifies the output linker definition filename for a Palm OS Cobalt Simulator target build.
- `-outSimgcc filename`
filename specifies the gcc-compatible output object file. When you use this option, the startup, stub and linker definition files generated are also generated as gcc-compatible files.
- `-outSimRsrc filename`
filename specifies the output acod resource file for a Palm OS Cobalt Simulator target build.
- `-outSimStartup filename`
filename specifies the output startup object filename for a Palm OS Cobalt Simulator target build.
- `-outSimStubs filename`
filename specifies the output stubs object filename for a Palm OS Cobalt Simulator target build.
- `-patchable [0 | 1]`
If you specify this optional parameter, it overrides the PATCHABLE keyword in the SLD file.
 - 0
By default, an exported function is unpatchable.
 - 1
By default, an exported function is patchable.

pslib issues a warning message when this command line option and the SLD file value are different.
- `-revision integer`
integer specifies a revision number.

If you specify this optional parameter, it overrides the REVISION keyword in the SLD file. *pslib* issues a warning message when this command line option and the SLD file value are different.

Using the Palm OS Shared Library Tool

Using the *pslib* Command Line Tool

-rsrcID *integer*

integer specifies a resource ID.

If you specify this optional parameter, it overrides the RESOURCEID keyword in the SLD file. *pslib* issues a warning message when this command line option and the SLD file value are different.

-type *four_character_code*

four_character_code specifies a 4-byte resource type.

If you specify this optional parameter, it overrides the type specification in the input SLD file. *pslib* issues a warning message when this command line option and the SLD file value are different.

-V

pslib displays the version information and exits.

NOTE: At least one output file (device target or Palm OS Cobalt Simulator target) must be specified or *pslib* issues an error message.

Using the Palm OS Post Linker

This chapter describes how you use `pelf2bin`, the Palm OS post linker, as part of the process of creating Palm OS applications.

- [“Palm OS Post Linker Concepts”](#) on page 44
- [“Using `pelf2bin` with Palm OS Developer Suite”](#) on page 45
- [“Using the `pelf2bin` Command Line Tool”](#) on page 45

Palm OS Post Linker Concepts

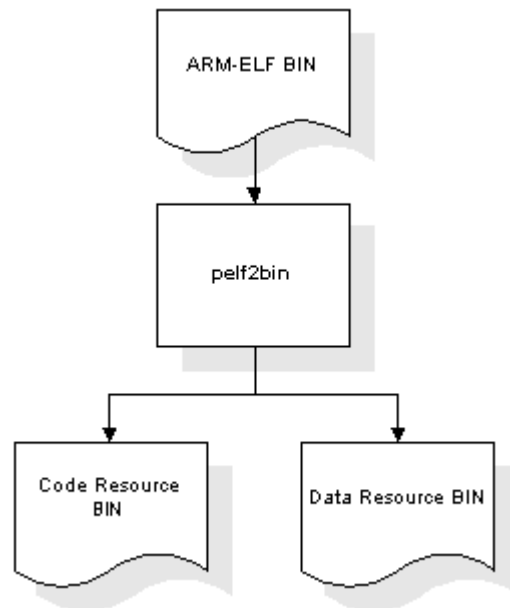
When you use an ARM-based compiler to compile your application source files, you create files in a standard ELF format. `pelf2bin` converts these ELF object files into binary resource files that can be merged into a Palm OS application.

`pelf2bin` extracts the code, data, and dynamic relocation sections from the input file, and produces two resource files:

- a file containing the compiled and linked code of the application
- a file containing the application's data and dynamic relocations, which is used by the Palm OS loader to prepare the application for execution

These files can be used with the resource tool `PRCMerge` to create a Palm OS application. For more information about `PRCMerge`, see the book *Palm OS Resource Tools Guide*.

Figure 6.1 Post Linker Overview



Using pelf2bin with Palm OS Developer Suite

pelf2bin is fully integrated with Palm OS Developer Suite. When you build your application or shared library with Palm OS Developer Suite, pelf2bin is called as part of the build process. You do not need to invoke pelf2bin directly.

Using the pelf2bin Command Line Tool

pelf2bin is used to convert an ELF object file into binary resource files that can be merged into a Palm OS application. The command line syntax for pelf2bin is:

```
pelf2bin [options] input_file
```

options

Additional command line options as described in the following section "[Specifying Command Line Options](#)."

input_file

Specifies the input ELF object file.

Specifying Command Line Options

-code *filename*

Specifies the code resource output filename.

filename

The default filename is `acod0000.bin`.

-data *filename*

Specifies the data resource output filename.

filename

The default filename is `adat0000.bin`.

-help

pelf2bin displays help information and ignores any other option.

-directory *dirname*

Specifies the output directory.

-rsrcID *value*

Specifies the resource ID.

Using the Palm OS Post Linker

Using the pelf2bin Command Line Tool

value

The default resource ID value is 0.

-verbose *level*

Specifies what level of diagnostic information you want **pelf2bin** to display.

level

An integer between 0 and 2.

-V

pelf2bin displays version information.

Shared Library Definition File Format Reference

This chapter provides reference information on the shared library definition (SLD) file format.

- “[Creating a Shared Library Definition File](#)” on page 48 describes the basic requirements for creating an SLD file.
- “[Specifying Keywords](#)” on page 48 describes the format of the keywords that you use in a SLD file.
- “[Sample Shared Library Definition Files](#)” on page 50 provides listings of sample SLD files.

Creating a Shared Library Definition File

Shared library definition (SLD) files are text files. You can use any text editor to create an SLD file. In the SDK samples, shared library files commonly have the filename extension `.sld`, but this extension is not required.

The file is arranged as a set of keyword/value pairs, separated with white space (with the exception of the `EXPORT` keyword as described below).

To add comments to an SLD file, use a semi-colon (`;`) character. If a line starts with a semi-colon, the entire line is treated as a comment. For any line that contains a semi-colon, `pslib` will ignore all of the characters that appear after the semi-colon character.

Specifying Keywords

Note that keywords are not case sensitive but the values specified are case sensitive.

`TYPE` *four_character_code*

Defines the type of the library.

`CREATOR` *four_character_code*

Defines the creator of the library.

`REVISION` *integer*

Specifies the revision number of the library

integer

A user-defined version non-negative number.

`RESOURCEID` *integer*

Specifies the resource ID of this library.

integer

A non-negative number. For PRC files that have more than one library, this value makes the libraries unique.

`ENTRY` *name*

Specifies the name of the entry point for this module. The `ENTRY` keyword is not required if your application's only entry point is the function `PilotMain()`.

PATCHABLE [0 | 1]

Defines the default patchability state for exported functions in the module.

0

By default, the library is unpatchable.

1

By default, the library is patchable.

OSVERSION *version*

Defines the minimum Palm OS version necessary to load this library. Use zero (0) if there is no minimum Palm OS version required.

ARMARCH [4T | 5T | 5TE | 0]

Defines the minimum ARM architecture necessary to load this library. Use zero (0) if there is no minimum ARM architecture required.

This value has meaning only for Palm OS device targets; it is not applicable for Palm OS Simulator targets.

EXPORTS *export_identifier*

Each line in the SLD file after the EXPORTS keyword defines a function name being exported. *export_identifier* is one of the following:

None

Indicates that there are no functions being exported.

name [*entry_id*] [*patch_indicator*]

Specifies a list of the names of the exported functions. An entry ID and patchability indicator may be associated with each function (separated by whitespace on the same line as the function name).

entry_id

If *entry_id* is specified, each function must have a unique entry ID. The function list must be sorted by entry point number, from 0 to *n*.

If you skip numbers in the list of entry points, the skipped entry points are treated as not implemented (or reserved) functions. For these reserved functions, `pslib` creates dummy functions. If such a function is

Shared Library Definition File Format Reference

Sample Shared Library Definition Files

called from a Palm OS application, Palm OS calls the `SysUnimplemented()` function.

patch_indicator

The patchability indicator has two values: `patchable` and `unpatchable`. If there is no patchability indicator, the default is defined by the `PATCHABLE` keyword or by `pslib`'s `-patchable` command-line option.

Sample Shared Library Definition Files

The section shows two sample SLD files:

- For a sample Palm OS application SLD file, see [Listing 7.1](#).
- For a sample Palm OS shared library SLD file, see [Listing 7.2](#).

Listing 7.1 Sample SLD File for an Application

```
;
; DateBook Library Definition File
;

TYPE                appl
CREATOR             dats
REVISION            1
RESOURCEID           0

ENTRY _PalmUIAppStartup
```

Listing 7.2 Sample SLD File for a Shared Library

```
;
; MathLib Library Definition File
;

TYPE                slib
CREATOR             math
REVISION            1
RESOURCEID           0

; Shared Libraries have one entry
ENTRY MathLibMain
```

Shared Library Definition File Format Reference

Sample Shared Library Definition Files

```
; Shared Library Exports List
```

```
EXPORTS
```

```
fabs
ceil
floor
rint
fmod
remainder
frexp
ldexp
modf
scalbn
exp
expm1
ilogb
log
log10
log1p
logb
cbrt
hypot
pow
sqrt
cos
sin
tan
cosh
sinh
tanh
acos
asin
atan
atan2
acosh
asinh
atanh
erf
erfc
lgamma
gamma
isnan
finite
copysign
nextafter
j0
j1
```

Shared Library Definition File Format Reference

Sample Shared Library Definition Files

```
jn  
matherr
```

Using elfdump

`elfdump` is a diagnostic tool that gives you information about the contents of an ELF object file.

NOTE: This tool is included in the compiler suite because you may find it useful, but it is an unsupported tool.

Using the elfdump Command Line Tool

`elfdump` reads the input ELF object files that you specify and generates a report of information about the ELF object files.

By default, `elfdump` output includes a header for each file and information for all sections in each file. But the command line options allow you to change the content and format of the output information.

Listing 8.1 Sample elfdump output

```
ELF FILE NAME: samplelib_startup.o
FILE CLASS:    32-bit objects
DATA ENCODING: little endian
FILE TYPE:     relocatable
ENTRY POINT:   undefined
TARGET:        ARM/Thumb Architecture
EABI VERSION:  2
ATTRIBUTES:
```

SECTION INFORMATION

section	offset	size	props	alignment	name
1	00000040	00000008	a + w	0004	runtime_helper_data_area
2	00000048	00000004	a + w	0004	palm\$\$slib_box1_0_0
.					
.					
.					

Using elfdump

elfdump Reference

```
    ____/ Section 1 \____
    /
    | name:      runtime_helper_data_area
    | type:      0x1 (progbits)
    | flags:     0x00000003 (allocated + writable)
    | address:   0x00000000
    | offset:    0x00000040
    | size:      0x8
    | link:      0x0
    | info:      0x0
    | alignment: 4
    |
00000000      00 00 00 00 00 00 00 00      . . . . .

    ____/ Section 2 \____
    /
    | name:      palm$$_slib_boxl_0_0
    | type:      0x1 (progbits)
    | flags:     0x00000003 (allocated + writable)
    | address:   0x00000000
    | offset:    0x00000048
    | size:      0x4
    | link:      0x0
    | info:      0x0
    | alignment: 4
    |
00000000      00 00 00 00      . . . .
```

elfdump Reference

This section provides reference information for the elfdump tool.

- [elfdump Command Line Interface](#)
- [elfdump Options](#)

elfdump Command Line Interface

The general format of the elfdump command line interface is this:

```
elfdump [options] input_files
```

options

elfdump options, as described in the section “[elfdump Options](#)” on page 55.

input_files

A list of ELF object files.

elfdump Options

- help
elfdump prints a summary of help.
- o *outputfile*
Sets the name of the output file to the name specified by *outputfile*.

If you do not specify an output filename, elfdump sends the output information to `stdout` (usually dumping the information to the screen).
- V
elfdump writes the its version numbers to `stderr`, and exits without performing any further actions.
- v *level*
Sets the elfdump verbosity level:
 - 0
elfdump displays errors only. This is the default verbosity level.
 - 1
elfdump displays warnings and errors.
 - 2
elfdump displays all messages.
- nodis
elfdump does not disassemble executable bytecode sections, instead showing them as hex data dumps.
- nodwarfdis
elfdump does not decode debug data.
- sortsyms
elfdump sorts the output symbol table by value.
- disdata
elfdump disassembles data sections as code including labels.
- allsyms
elfdump shows all (possibly superfluous) symbols in the disassembly.

Using elfdump

elfdump Reference

-arch *vers*

elfdump disassemble for the given instruction set architecture.

vers

An instruction set architecture value.

Valid values: v3, v3M, v4, v4xM, v4T, v4TxM, v5, v5xM, v5T, v5TxM, v5TexP, v5TE

The default value is v5TE.

-summary

elfdump includes only segment and section summaries.

Index

Symbols

- 55

A

- adding file to library 30
- adding local symbols to output 26
- ARM architecture support 39
- assembler options 22

B

- building a Palm OS application 1

C

- C language standard 12
- C++ language standard 12
- C++ symbols 26
- changing search order 14
- compiler
 - options 13
 - overview 8
 - see also pacc 8
- compiler chain 8
- compiler search order 14
- compiler tools 8
- compiling without linking 13
- creator ID for shared library 39

D

- debug information 24
- defining names 13
- developer tools vi
- diagnostic tool
 - overview 10
 - see also elfdump 10
- disabling exception handling 16
- displaying library symbol table 33
- displaying logo banner 15
- DLL name 40
- documentation vii

E

- elfdump

- description 53
- options 55
- overview 10
- reference 54

- enabling exception handling 14
- enforcing strict ANSI rules 18
- entry point 2, 24, 40
- error message output 26
- error messages 20
- exception handling, disabling 16
- exception handling, enabling 14
- executable name 40
- extracting file from library 31

F

- file map 26

H

- hiding logo banner 16

I

- including symbolic debugging information 14
- inline functions 14
- inlining functions 14

K

- knowledge base viii

L

- librarian
 - overview 9
 - see also palib 9
- library path 15
- library search path 25
- linker help information 24
- linker information 25
- linker options in a file 27
- linker symbols 26
- listing files in library 31
- local symbol output 26
- logo banner 15

N

-nostackwarn 16

O

omitting debug information 26

omitting local symbols 26

optimization level 16

ordering output sections 25

output file 22

output file map 26

output file name 16

output filename 26

output linker symbols 26

P

paasm

description 21

options 22

-o 22

-V 23

pacc 11

command line interface 12

description 12

options 13

-C 13

-c 13

-D 13

-E 14

-ex 14

-g 14

-g0 14

-I 14

-L 15

-logo 15

-noex 16

-nologo 16

-O 16

-o 16

-P 17

-S 18

-strict 18

-U 18

-V 18

-v 19

-vv 19

-w 19

-Werror 20

-Wstrict 20

overview 8

palib

adding file 30

description 29

displaying file list 31

displaying symbols 33

extracting file 31

options 33

-add 30

-delete 30

-extract 31

-replace 30

-syntab 33

-toc 31

-V 34

-via 34

overview 9

reference 32

removing file 30

replacing file 30

palink

description 24

options 24

-d 24

-entry 24

-errors 24

-first 25

-help 24

-info 25

-libpath 25

-list 26

-locals 26

-mangled 26

-map 26

-nodebug 26

-nolocals 26

-o 26

-symbols 26

-unmangled 26

-V 27

-via 27

-xref 27

palink debug information 24

Palm OS assembler 21

- see also paasm 21
- Palm OS compiler
 - see Palm OS Protein C/C++ Compiler 11
- Palm OS Developer Suite vi
- Palm OS librarian 29
 - see also palib 9
- Palm OS linker 24
 - see also palink 24
- Palm OS Protein C/C++ Compiler 11, 12
- pelf2bin
 - options
 - V 46
- preinclude= 17
- preprocessing source 14
- preprocessing source files 17
- producing assembly files 18
- pslib 36
 - options
 - ARMarch 39
 - creator 39
 - entry 40
 - execName 40
 - V 42

R

- redirecting error messages 24
- redirecting output messages 26
- removing file from library 30
- replacing file in library 30
- retaining comments 13

S

- search order 14
- setting library search path 25
- shared library definition 2
- shared library tool
 - overview 36

- sld file 2
- source equivalents 26
- specifying ARM architecture 39
- specifying creator ID 39
- specifying entry point 24, 40
- specifying executable name 40
- specifying linker options 27
- specifying output filename 26
- stackwarn 18
- strict ANSI rules 18, 20
- suppressing warning messages 19
- symbolic debugging information 14

T

- tools documentation vi
 - Introduction to Palm OS Tools vi
 - Language and Library Reference vi
 - Palm OS Debugger vi
 - Palm OS Resource Editor vi
 - Resource Tools vii
 - Virtual Phone vii
- training vii

U

- undefining names 18
- using a file for options 34
- using file for linker options 27

V

- verbose output 19
- verbose output with no execution 19
- version information 23, 27
- version string 18, 34, 46

W

- warning messages 19

