

Solaris Compatibility Libraries for Compaq Tru64 UNIX

User's Guide

March 2000

Product Version: Solaris Compatibility Libraries Version 1.1

Operating System and Version: Compaq Tru64 UNIX Version 4.0D or
higher

This manual describes how to install and use the Solaris Compatibility
Libraries

All of the documentation and software included in the Solaris Compatibility Libraries for Tru64 UNIX is copyrighted by Compaq Computer Corporation

Copyright (c) 2000 Compaq Computer Corporation.

All rights reserved. Unpublished rights reserved under copyright laws of the United States of America.

Possession, use, duplication or dissemination of the software and media source and binary forms, with or without modification, is authorized provided that the following conditions are met:

1. Redistributions of source code and its associated documentation must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of the Corporation may not be used to endorse or promote products derived from this software without specific prior written permission from Compaq Computer Corporation.

THIS SOFTWARE IS PROVIDED BY COMPAQ COMPUTER CORPORATION "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE CORPORATION BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

A small portion of the SCL code is derived from BSD sources, and we therefore reproduce the following notice:

Copyright (c) 1990, 1993 The Regents of the University of California. All rights reserved.

This code is derived from software contributed to Berkeley by Chris Torek.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by the University of California, Berkeley and its contributors.
4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The following notice is reproduced from the Sun RPC distribution:

Sun RPC is a product of Sun Microsystems, Inc. and is provided for unrestricted use provided that this legend is included on all tape media and as a part of the software program in whole or part. Users may copy or modify Sun RPC without charge, but are not authorized to license or distribute it to anyone else except as part of a product or program developed by the user.

SUN RPC IS PROVIDED AS IS WITH NO WARRANTIES OF ANY KIND INCLUDING THE WARRANTIES OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, OR ARISING FROM A COURSE OF DEALING, USAGE OR TRADE PRACTICE.

Sun RPC is provided with no support and without any obligation on the part of Sun Microsystems, Inc. to assist in its use, correction, modification or enhancement.

SUN MICROSYSTEMS, INC. SHALL HAVE NO LIABILITY WITH RESPECT TO THE INFRINGEMENT OF COPYRIGHTS, TRADE SECRETS OR ANY PATENTS BY SUN RPC OR ANY PART THEREOF.

In no event will Sun Microsystems, Inc. be liable for any lost revenue or profits or other special, indirect and consequential damages, even if Sun has been advised of the possibility of such damages.

Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, California 94043

Sun, Sun Microsystems, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

SPARC is a trademark or registered trademark of SPARC International, Inc. in the United States and other countries.

COMPAQ, the Compaq logo, and the Digital logo are registered in the U.S. Patent and Trademark Office. Alpha, AlphaServer, NonStop, TruCluster, and Tru64 are trademarks of Compaq Computer Corporation.

Microsoft and Windows NT are registered trademarks of Microsoft Corporation. Intel, Pentium, and Intel Inside are registered trademarks of Intel Corporation. UNIX is a registered trademark and The Open Group is a trademark of The Open Group in the United States and other countries. Other product names mentioned herein may be the trademarks of their respective companies.

Possession, use, or copying of the software described in this publication is authorized only pursuant to a valid written license from Compaq Computer Corporation or an authorized sublicensor.

Compaq Computer Corporation shall not be liable for technical or editorial errors or omissions contained herein. The information in this document is subject to change without notice.

Contents

1 Getting Started

| | | |
|-------|--|-----|
| 1.1 | Introduction | 1-1 |
| 1.2 | Assumptions | 1-2 |
| 1.3 | Installation and Content of the SCL | 1-3 |
| 1.3.1 | The Solaris Compatibility Libraries Directory Tree | 1-3 |
| 1.3.2 | Installing the SCL | 1-4 |
| 1.3.3 | Header Files | 1-5 |
| 1.3.4 | Libraries | 1-6 |
| 1.3.5 | Binaries | 1-7 |
| 1.3.6 | Documentation | 1-7 |

2 Remote Procedure Calls

| | | |
|---------|--|-----|
| 2.1 | Background | 2-1 |
| 2.2 | Transport Independent RPC | 2-1 |
| 2.3 | Portmapper Functionality | 2-2 |
| 2.3.1 | The portmap Program | 2-2 |
| 2.3.2 | rpcbind | 2-2 |
| 2.3.2.1 | Specifying the TCP/UDP Port Number | 2-2 |
| 2.4 | Executable Programs Supplied with SCL RPC | 2-3 |
| 2.4.1 | rpcgen | 2-3 |
| 2.4.2 | rpcinfo | 2-4 |
| 2.4.3 | rpcbind | 2-4 |
| 2.5 | Setting up SCL RPC | 2-4 |
| 2.5.1 | Configure the XTI kernel component | 2-5 |
| 2.5.2 | Create the scl_netconfig File | 2-5 |
| 2.5.3 | Edit /etc/services | 2-6 |
| 2.5.4 | Ensure the loader Can Locate the Shared Libraries | 2-6 |
| 2.5.5 | Run rpcbind | 2-6 |
| 2.6 | Building Applications that use the SCL RPC Libraries | 2-6 |
| 2.6.1 | Square Example | 2-7 |
| 2.7 | Message Logging | 2-8 |
| 2.8 | Known Problems | 2-8 |
| 2.9 | Restrictions | 2-8 |

| | | |
|----------|---|------|
| 3 | Solaris Threads | |
| 3.1 | Background | 3-1 |
| 3.2 | STL Functionality | 3-2 |
| 3.3 | Components of the Solaris Thread Library | 3-3 |
| 3.4 | Building Applications Against STL | 3-3 |
| 3.5 | STL Environmental Settings | 3-4 |
| 3.6 | Overview of the STL Implementation | 3-6 |
| 3.6.1 | Data types | 3-6 |
| 3.6.2 | Functions | 3-6 |
| 3.6.3 | Performance | 3-6 |
| 3.7 | Issues When Running Solaris Thread Programs | 3-7 |
| 3.7.1 | Displaying a Thread's Identifier | 3-7 |
| 3.7.2 | Status Returns from Functions | 3-8 |
| 3.7.3 | STL Aborts on Detecting an Internal Error | 3-9 |
| 3.7.4 | Incautious Use of thr_suspend() and thr_continue() Can Hang an Application | 3-10 |
| 3.7.5 | Joining threads | 3-10 |
| 3.7.6 | Suspending and Continuing a Thread Sometimes Fails ... | 3-11 |
| 4 | Miscellaneous Components | |
| 4.1 | Signal Name Library Routines | 4-1 |
| 4.2 | Directory Path Functions | 4-2 |
| 4.3 | High-resolution Timers | 4-2 |
| 4.4 | Asynchronous I/O Library Routines | 4-3 |
| 4.5 | Large File Support | 4-3 |
| 4.6 | Text Domain Library Routines | 4-4 |
| 4.6.1 | Documentation | 4-5 |
| 4.6.2 | Installing GNU gettext Tool | 4-5 |
| 4.6.3 | Using GNU gettext Tool | 4-6 |
| 4.7 | Wide-character Support | 4-6 |
| 4.8 | Runtime Dynamic Linking | 4-7 |
| 4.9 | DNS Resolver Library Routines | 4-8 |
| 4.10 | Remote User Library Routines | 4-8 |
| 4.11 | Support for tell(), snprintf() and vsnprintf() Library Routines | 4-9 |
| 4.12 | Support for makedev() Function | 4-9 |
| 4.13 | File-related Differences | 4-10 |

5 Error Logging

6 Reference Pages

7 Building the SCL Source Code

| | | |
|-----|--|-----|
| 7.1 | Unpacking the Source Files | 7-1 |
| 7.2 | Building the Source Tree | 7-2 |
| 7.3 | Testing the SCL Run-Time Directories | 7-3 |
| 7.4 | Using the SCL Run-Time Directories | 7-3 |
| 7.5 | Redistributing the Run-Time and Source Directories | 7-4 |

A Mapping Solaris thread types to POSIX thread types by STL

B Solaris thread functions implemented by STL

C SCL Installation Example

Index

Figures

| | | |
|-----|--------------------------------|-----|
| 1-1 | SCL Root Directory | 1-4 |
| 7-1 | SCL Source Directories | 7-2 |
| 7-2 | SCL Run-time Directories | 7-3 |

Tables

| | | |
|-----|--|-----|
| 1-1 | Subsets for Software Development | 1-2 |
| 1-2 | SCL Subsets | 1-4 |
| 3-1 | Converting from Solaris threads to POSIX threads | 3-1 |
| 3-2 | STL Functionality | 3-2 |
| 3-3 | POSIX Threads Data Types | 3-6 |
| 3-4 | Return Status from condition-wait Functions | 3-8 |

Getting Started

This chapter introduces the Solaris Compatibility Libraries for Tru64 UNIX and describes how to install it.

1.1 Introduction

In addition to the standard UNIX application program interfaces (APIs) common to most UNIX system, the Sun Solaris operating system provides a number of APIs specific to Solaris. An application that uses these APIs might require substantial changes before it can be compiled on a UNIX system other than Solaris.

The Solaris Compatibility Libraries (SCL) is a collection of C header files and libraries that implement many of the key APIs specific to Solaris. The purpose of the SCL is to enable Solaris applications that call these non-standard APIs to be built and executed on Tru64 UNIX with minimal source code changes.

The functionality provided by the SCL is split into three main categories:

- Remote Procedure Calls
- Solaris Threads
- Miscellaneous Components

These are described in chapters later in this guide.

The SCL does not attempt to resolve general platform-specific differences that are of concern when porting applications from Solaris to Tru64 UNIX, such as the following:

- Sun Solaris on SPARC stores data in big-endian format; Tru64 UNIX on Alpha stores data in little-endian format. Solaris programs which assume big-endian data-storage may build cleanly on Tru64 UNIX, but may not produce the desired results at run-time.
- Addresses (pointers) and the C data-type "long" are 64-bit on Tru64 UNIX. While recent versions of Sun Solaris support both 32-bit and 64-bit address environments, it is expected that many Solaris applications were written at a time when Solaris provided only 32-bit addresses. SCL does not attempt to resolve 32-bit to 64-bit porting issues.

The Porting Assistant utility from Compaq can help identify these types of problems. Porting Assistant ships with the Developers' Toolkit for Tru64 UNIX Version 4.0 and later. The subset name for Porting Assistant is `PRTBASE nnn` , where the value of nnn depends on the version of Porting Assistant. You can download the Porting Assistant from http://www.digital.com/info/porting_assistant/.

The *Sun Solaris to Compaq Tru64 UNIX Porting Guide* presents a range of issues involved in porting from Solaris to Tru64 UNIX. The guide can be viewed online or downloaded at the following URL:

http://www.unix.digital.com/faqs/publications/pub_page/porting.html

1.2 Assumptions

It is assumed that you have an Alpha system running Tru64 UNIX Version 4.0D or higher. If you are building applications that use the SCL, the system must have the software development tools installed. In particular, the following subsets must be installed (nnn in the subset name represents the version number, which varies according to which release of Tru64 UNIX you have installed.):

Table 1–1: Subsets for Software Development

| Software Subset | Content |
|---|--|
| <code>OSFSDEnnn</code> | Software Development Tools and Utilities |
| <code>OSFPGMRnnn</code> | Standard Programmer Commands |
| <code>OSFINCLUDEnnn</code> | Standard Header Files |
| <code>OSFLIBAnnn</code> | Static Libraries |
| <code>OSFCMPLRSnnn</code> | Compiler Back End |

See `setld(1)` for information on using the `setld` command to install software subsets.

It is assumed that the developers using these libraries and the associated documentation are familiar with the Solaris API and so are already familiar with the functionality provided by the SCL.

Sun documentation can be viewed online at <http://docs.sun.com>.

Tru64 UNIX documentation can be viewed online at the following URL:

http://www.unix.digital.com/faqs/publications/pub_page/pubs_page.html

1.3 Installation and Content of the SCL

This section describes how to install the SCL on your Tru64 UNIX system. The content of the SCL kit and the locations of installed files are presented.

1.3.1 The Solaris Compatibility Libraries Directory Tree

The SCL is installed using the `setld` software subset management utility that is supplied with Tru64 UNIX. Consequently the software is installed beneath a directory in `/usr/opt` based on the version number of the software. Version 1.1 will be installed into directory `/usr/opt/SCL110`.

In order to provide a standard mechanism for referencing the latest version of SCL from application build files and commands, you can create a symbolic link in `/usr/opt` to reference an SCL directory. This link is `/usr/opt/solcomplib`. This location is thus known as the SCL root directory, and application `make` files and build commands should use this with the `-L` and `-I` compiler switches. If you install newer or alternative versions of SCL, you can move the `solcomplib` link to reference this new version of SCL, thus requiring no changes to build files and commands.

Although it is strongly recommended that you use the `/usr/opt/solcomplib` link to reference the SCL root directory, it is possible to create an alternative link or to directly reference the installation directory. In this case, you must set the `SCL_ROOT_DIR` environment variable to reference the alternative SCL root directory name.

Figure 1–1 shows the directory structure for the SCL.

Installation of the `SCLBASE` subset creates the entire SCL root directory tree as shown in Figure 1–1, with the exception of the `doc`, `man`, and `source` directories. These are created by installation of the `SCLDOC` and `SCLSOURCE` subsets.

Installation of the `SCLSOURCE` subset places a `tar` file called `sclsource.tar` in the `source` directory beneath the SCL root directory. The `tar` file can be unpacked into any location and creates a source tree that can be used to modify and build the entire SCL. See Chapter 7 for additional information on using and building the SCL source code.

Installation of any subset places the respective components of SCL in a directory beneath `/usr/opt` named `SCLnnn`. Version 1.1 of SCL therefore resides in `/usr/opt/SCL110`.

Follow these steps to install the SCL:

1. If SCL version 1.0 has previously been installed, then either delete or rename the directory `/usr/opt/solcomplib`. This enables successful creation of the symbolic link during the installation of version 1.1
2. Unpack the SCL distribution to create the subsets by entering the following commands:

```
# mkdir directory/sclkit
# mv downloaded-compressed-tar-file directory/sclkit
# cd directory/sclkit
# gunzip SCL110.tar
# tar xvf SCL110.tar
```
3. Use `setld` to install the selected subsets. If you omit the subset names, `setld` prompts for the names of subsets to install.

```
# setld -l . SCLBASEnnn SCLDOCnnn SCLSOURCEnnn
```
4. The `SCLBASE` subset has a post-installation step that asks if a script named `install` should be run. This script performs post-installation tasks. The only task performed is to create a symbolic link to the SCL message catalog in the appropriate system directory as indicated by the current value of the `LANG` environment variable. If `LANG` is not defined for the process performing the installation, the script gives a reminder to define `NLSPATH`.

An example of an SCL installation appears in Appendix C

1.3.3 Header Files

The Solaris Compatibility Libraries provides numerous C language headers that define types and structures as well as the Solaris function prototypes. Header files are located in the `include` directory beneath the SCL root directory.

To use the SCL versions of the headers, your `make` files and build commands must specify the SCL `include` directory before any other directories.

Many of the SCL headers provide prototypes and definitions that are in addition to those in the standard headers. Consequently, these SCL headers include the standard headers by specifying an explicit path. The standard headers are assumed to reside in the `/usr/include` directory tree. If your Tru64 UNIX development environment differs from this, you must edit the SCL headers and change the directory paths to match those in your environment.

1.3.4 Libraries

The SCL functionality is provided as a number of shared libraries residing in the `shlib` directory beneath the SCL root directory. To use shared libraries, you must set the environment variable `LD_LIBRARY_PATH` to include the SCL `shlib` path so that the SCL shared libraries can be located at run-time.

In addition, the SCL miscellaneous components are supplied as a static library residing in the `lib` directory.

When building applications that use the SCL it is important to name the appropriate SCL directory with `-L` option on the `cc` or `ld` command, or within the makefile. The libraries to specify are:

- `-lrpc -lxti` for RPC
- `-lthread` for threads
- `-lsolmisc` for miscellaneous functions

If linking with `libsolmisc.a` (the static library), you need to also specify additional libraries: `-lsolutil` and possibly `-lrt`, `-lexc`, and `-lrpcsvc`

The shared libraries must be installed on every system that runs applications that make use of the SCL. You should either install the SCLBASE subset on these systems (as detailed previously) or package all required SCL libraries with your application distribution.

For details on how to link SCL RPC applications, see the `square` example in the `examples/rpc` directory beneath the SCL root directory.

For details of how to link SCL thread applications, see Section 3.4 and the example program `stl_example` in the `examples/threads` directory beneath the SCL root directory.

1.3.5 Binaries

A number of commands and utilities are supplied with the SCL and these reside in the `bin` directory beneath the SCL root directory. You should amend the value for `PATH` environment variable in order to locate these.

One of the utilities supplied in the `bin` directory, `tru64_version`, may be used with build commands to define a preprocessor symbol that contains the version of Tru64 UNIX on which the build is occurring. Such a definition is useful in circumstances where conditional compilation is required to distinguish functionality on different versions of Tru64 UNIX.

An example of this can be found in the SCL-supplied `<sys/types.h>` header for types `t_scalar_t` and `t_uscalar_t`. These types exist on Tru64 UNIX Version 5.0 and later but not on Version 4, and so the header conditionally defines them based on the value of the `TRU64_VERSION` preprocessor symbol.

Applications that include this header should ensure that the preprocessor symbol is defined by adding the `-D`tru64_version -m`` option to the `cc` command. The command `tru64_version -?` lists all features of this utility.

1.3.6 Documentation

You can find documentation provided with SCL in the `doc` directory beneath the SCL root directory. Documentation comprises this User Guide and reference pages for all of the functions and commands implemented by SCL. The reference pages are available as man pages and in HTML format. Refer to Chapter 6 for additional information on reference documentation.

Remote Procedure Calls

This chapter contains information about installing and using the RPC component of the Solaris Compatibility Libraries for Tru64 UNIX (SCL).

2.1 Background

This implementation is based on a port of SunSoft's transport-independent RPC (TI-RPC) v2.3, the distribution of which is publicly available at <ftp://playground.sun.com/pub/rpc>. Refer to the README file included in that distribution for information describing the release.

The Tru64 UNIX operating system ships with a version of ONC RPC that is derived from an earlier distribution of code from Sun. This version is sockets-based and is documented in the manual *Programming with ONC RPC*, which can be found in the Programming Section of the Tru64 UNIX documentation.

Throughout this document, the term "SCL RPC" refers to the TI-RPC library that is part of the SCL, and "Sockets RPC" to the RPC implementation that is shipped as part of the Tru64 UNIX distribution.

This chapter assumes that the SCL has been installed and that the SCL root directory is referenced by the standard symbolic link `/usr/opt/solcomplib`.

It is assumed that users of SCL RPC will be familiar with TI-RPC v2.3 or later on Solaris systems.

2.2 Transport Independent RPC

Sockets RPC uses the sockets API to communicate between client and server using either TCP or UDP.

The SCL RPC library uses the X/Open Transport Interface (XTI) API. This means that communication between client and server is not necessarily restricted to TCP or UDP protocols, but could potentially make use of any available transport provider. However, the SCL RPC library supports only TCP and UDP transports.

2.3 Portmapper Functionality

Before an RPC server can run on a host, that host must already be running a portmapper program. It is the job of the portmapper to keep track of which RPC servers are running.

Every RPC server that starts up must inform the portmapper on the local node that it is ready to accept client requests.

When an RPC client application wishes to make a call to a server application on a remote node, it will first communicate with the portmapper on that node, and the portmapper will return information to the client about the availability (or not) of the requested service.

2.3.1 The portmap Program

For Sockets RPC, the portmapper functionality is provided by the `portmap` program. The `portmap` program maintains a list of which servers are running, and which TCP/UDP ports they are listening on.

Applications using Sockets RPC always use port 111 to communicate with the `portmap` program (which itself is hardcoded to bind to port 111). It is not possible to make such applications use a different port number.

2.3.2 `rpcbind`

For SCL RPC, the `portmap` program is replaced by the `rpcbind`, which provides essentially the same functionality, but uses the TI-RPC libraries.

Because the `rpcbind` program replaces `portmap` on Solaris systems, it would normally expect to use port 111 for TCP and UDP connections; it is not intended that both `portmap` and `rpcbind` will be running at once.

When running SCL RPC, the port number used by components for TCP and UDP connections is determined by an entry in the `/etc/services` file. This means that you can configure SCL RPC to run using a port number other than 111, and so avoid any conflicts with existing applications that use Sockets RPC.

2.3.2.1 Specifying the TCP/UDP Port Number

To assign a port number for TCP and UDP based connections, edit `/etc/services` and assign the desired `portnumber/protocolname` to the service named `scl_rpcbind`.

For example, the following lines in `/etc/services` force `rpcbind` to use port 1112 for UDP and TCP connections:

```
scl_rpcbind 1112/udp # Use port 1112 for SCL RPC library
scl_rpcbind 1112/tcp #
```

For SCL RPC applications to work, an entry for the `scl_rpcbind` service must exist; the library will not supply a default value if no entry is found.

The effects of specifying an entry for the `scl_rpcbind` service are as follows:

- The `rpcbind` program attempts to bind to the port specified by `scl_rpcbind`.
- SCL RPC server applications perform a lookup on `scl_rpcbind` and use the resultant port number to communicate with `rpcbind` on the local host
- SCL client applications perform a lookup of `scl_rpcbind` and use the resultant port number to communicate with `rpcbind` on any remote host.

The implication of this is that if a port number other than 111 is used for SCL RPC on a given host, then RPC communication between that host and another host in the network will only work if the other host has been also been configured to use the alternate port number.

2.4 Executable Programs Supplied with SCL RPC

The `bin` directory beneath the SCL root directory (`/usr/opt/solcomplib/bin` by default) contains three executable programs for use with the SCL RPC component. These executables make use of the SCL libraries, and so require that the loader be able to locate the SCL shared libraries at run-time (see Section 1.3.4).

The programs described in this section are ports of the utilities as included in the public distribution of SunSoft's TI-RPC v2.3 implementation.

2.4.1 `rpcgen`

The new `rpcgen` utility is an enhanced version of the `rpcgen` program that ships as part of Sockets RPC with Tru64 UNIX. This version of `rpcgen` provides various new functionality, including (the following is taken from the README files from the SunSoft distribution):

- multiple arguments
- argument pass by value
- client and server template files, makefile template file
- ANSI C RPC headers
- ANSI C code generated for stubs and XDR functions

- `xdr_inline` for structures with more than 3 elements for better performance

2.4.2 rpcinfo

The new `rpcinfo` utility is an enhanced version of the `rpcinfo` program that ships as part of Sockets RPC with Tru64 UNIX. This version of `rpcinfo` is compatible with the `portmap` program as well as `rpcbind`. Note that `rpcinfo` uses the `scl_rpcbind` entry in `/etc/services` to determine which port number to use to contact a portmapper daemon.

In order to use the new version of `rpcinfo` to communicate with the standard `portmap` program, the entry for `scl_rpcbind` in `/etc/services` must specify a port number of 111.

2.4.3 rpcbind

The new `rpcbind` utility is used by SCL RPC in place of the old `portmap` utility. See Section 2.3.2.

2.5 Setting up SCL RPC

To use the SCL RPC library, you must first perform the following tasks. The rest of this section describes these tasks in detail.

1. Configure the XTI kernel component
2. Create the `scl_netconfig` file
3. Edit the `/etc/services` file, adding an entry for the service `scl_rpcbind`.
4. Ensure that the loader can locate the SCL shared libraries
5. Run `rpcbind` (if required)

The executable program `test_sclrpc_config` is supplied in the `examples/rpc` directory beneath the SCL root directory. You can use `test_sclrpc_config` to check the configuration of a system for use with SCL RPC. The following is an example run of this program:

```
# test_sclrpc_config
Testing SCL RPC configuration
-----

Testing for service name 'scl_rpcbind'....OK

Testing for existence of scl_netconfig file....OK

Looking for SCL RPC shared objects....OK
```

```

Testing APIs for scl_netconfig file..
.found TCP entry..
.found UDP entry..
.Found 2 valid entries in scl_netconfig file

Testing access to XTI devices..
.succeeded in opening /dev/streams/xtiso/tcp+
.succeeded in opening /dev/streams/xtiso/udp+
. 2 XTI devices OK

Total errors found : 0

```

2.5.1 Configure the XTI kernel component

SCL RPC accesses the network through the X/Open Transport Interface (XTI). Therefore, any system on which you wish to use SCL RPC must have the `xtiso` kernel option installed. See *Configuring XTI Transport Providers* in the *Tru64 UNIX Network Programmer's Guide*.

2.5.2 Create the `scl_netconfig` File

Solaris supplies the `/etc/netconfig` file, which is a text file containing one line for each supported XTI protocol. Solaris also provides various functions, such as `getnetconfig` and `setnetconfig`, which make use of the data in this file. Various components in the SCL RPC library make use of this set of functions, and consequently expect such a file to exist.

The SCL RPC library looks for a file called `scl_netconfig` to satisfy this requirement. This file should be located in the `etc` directory beneath the SCL root directory; for example, `/usr/opt/solcomplib/etc`.

The `scl_netconfig` file has the same format as the corresponding Solaris file, although the contents of the file will be different. The following is an extract of a typical `scl_netconfig` file:

```

#nw-id semantics flags family name device nametoaddr_libs
#-----
#
udp tpi_clts v inet udp /dev/streams/xtiso/udp+ sclrpcsupport.so
tcp tpi_cots_ord v inet tcp /dev/streams/xtiso/tcp+ sclrpcsupport.so

```

Note that the `device` field refers to the devices that will have been installed as a result of configuring the kernel `xtiso` option, and that the `nametoaddr_libs` field should contain the name of the library `sclrpcsupport.so`, which is supplied as part of the SCL distribution.

Note also that there should be no loopback devices specified in this file; these are not supplied with Tru64 UNIX.

A sample `scl_netconfig` file can be found in the `examples/rpc` directory beneath the SCL root directory.

2.5.3 Edit /etc/services

The SCL RPC components require that the `/etc/services` file has entries for `scl_rpcbind`. See Section 2.3.2.1.

2.5.4 Ensure the loader Can Locate the Shared Libraries

The SCL RPC library code is contained in the shared object `librpc.so`, which should be included in any link operation for SCL RPC applications. This library, along with the other shared objects supplied as part of the SCL package, is located in the `shlib` directory beneath the SCL root directory, `/usr/opt/solcomplib/shlib` by default.

At runtime, the loader must locate the SCL shared library files. In order for the loader to be able to locate these files, you can set the `LD_LIBRARY_PATH` variable. For example, to set the variable in C shell:

```
# setenv LD_LIBRARY_PATH /usr/opt/solcomplib/shlib
```

For more information, see the Tru64 UNIX reference page for `loader(5)`.

2.5.5 Run rpcbind

The `rpcbind` program can be started only by a process with superuser privileges. Because it uses the SCL libraries, the `LD_LIBRARY_PATH` environment variable should be defined to point to the appropriate directory before starting `rpcbind`.

By default, `rpcbind` forks and runs as a daemon process. To prevent this, use the `-d` command line option. This option also causes `rpcbind` to display additional diagnostic information to `stdout`.

Note that `rpcbind` binds to the port number specified for the `scl_rpcbind` service in `/etc/services`. For more information, see Section 2.3.2.

2.6 Building Applications that use the SCL RPC Libraries

The `examples/rpc` directory beneath the SCL root directory contains a sample application that can be built to use the SCL RPC libraries, as well as a file that defines various make macros that are useful for building SCL RPC applications.

To build a client and server application, do the following:

1. Use `rpcgen` to process the application's RPC specification file(s)
2. Compile the client/server application components
3. Link the client/server application components
4. Execute the server application

5. Execute the client application

All of these steps can be performed with Sockets RPC or SCL RPC components.

It is also possible to use the Sockets RPC components for some of the steps in this process, and SCL RPC for later steps. For example, it may be considered appropriate to take existing object files that were built from code that was generated with the old (Sockets RPC) version of `rpcgen`, and link them against the SCL RPC libraries.

Note that it would not be possible to take code generated by the newer version of `rpcgen` and link that against the older (Sockets RPC) version of the libraries.

Because SCL RPC uses the XTI interface, applications linking with the SCL RPC library must also link with the `xti` library (`libxti`). You can do this by including `-lxti` in the `ld` command line.

2.6.1 Square Example

The directories `dir`, `msg`, `sort`, and `square` beneath the `examples/rpc` directory contain example RPC applications that you can build to use either Sockets RPC or SCL RPC.

The makefile for each example includes the file `SCL_RPC_Make-macros` (located in `examples/rpc`), which allows you to build the application in any of the following ways:

- Building with `CONFIG=NATIVE` causes the application to be built using Sockets RPC. No use is made of any SCL RPC components:

```
# make CONFIG=NATIVE all
```
- Building with `CONFIG=SCL` causes the application to be built using SCL RPC, with `rpcgen` being used in its default mode, and the C compiler being invoked with `-std0` (K & R mode):

```
# make CONFIG=SCL all
```
- Building with `CONFIG=SCL_ANSI` causes the application to be built using SCL RPC, with `rpcgen` being used to generate ANSI format source code, and with the C compiler being invoked with `-std1` (ANSI mode):

```
# make CONFIG=SCL_ANSI all
```

For both `CONFIG=SCL` and `CONFIG=SCL_ANSI`, `make` macros are defined that cause the correct include paths to be used, and the correct libraries to be linked.

The command `make clean` can be used to remove all but the original source files from the appropriate directory.

2.7 Message Logging

The SCL RPC components perform message logging to the file specified by the environment variable `SCL_LOG_FILE`, if it is defined. For more information, see Chapter 5.

In addition, the TI-RPC code uses the `syslog()` library routine to log messages for certain error situations. Specifically, detached (daemon) processes, such as `rpcbind`, or SCL RPC server applications may log messages to the `daemon.log` logfile. For more information, see the Tru64 UNIX reference page for `syslogd(8)`.

2.8 Known Problems

The following is a known problem in SCL Version 1.0.

- SCL RPC server applications might fail to register themselves with `portmap`.

In some situations, SCL RPC server applications may not be able to register themselves with the `portmap` utility (as opposed to `rpcbind`). This may happen, for example, if the `scl_rpcbind` entry in `/etc/services` specifies a port number of 111, and `portmap` is used instead of `rpcbind`.

In these situations, a call to `svc_create` returns zero, and a message is logged. See Chapter 5.

2.9 Restrictions

The following restrictions apply to SCL RPC:

- Long double datatype is not supported.

The SCL RPC library does not provide XDR support for long double (`xdr_quaduple`) data types.

- The SCL RPC library supports only TCP and UDP transports.
- Avoid using port 111 for SCL RPC.

The SCL RPC libraries have not been fully tested with `scl_rpcbind` set to port 111. It is recommended that you use the standard `portmap` program with port 111, and that you allocate a different port number for SCL RPC use.

- The `librac` library is not supported.
- Secure RPC is not supported.
- Note that TI-RPC v2.3 does not provide support for multi-threaded server applications; this functionality is not supported until TI-RPC v2.4

Solaris Threads

The Solaris Thread Library (STL) for Tru64 UNIX provides an implementation of most of the Solaris threads application programming interface (API) for Tru64 UNIX. The objective is to make the task of porting applications which use the Solaris thread API to Tru64 UNIX be as easy as a re-compile.

3.1 Background

In 1995 the IEEE defined a POSIX standard, IEEE POSIX 1003.1c-1995, for a threads API, which we refer to as POSIX threads (also commonly referred to as pthreads). Most operating systems now support POSIX threads, sometimes instead of their proprietary thread interface (as with DEC's CMA, and sometimes in addition to the proprietary interface (as with Sun's Solaris threads).

For maximum portability, a threaded program should use the POSIX threads API. To convert an application from the Solaris threads API to the POSIX threads API may mostly be a matter of making some editor substitutions, as shown below:

Table 3–1: Converting from Solaris threads to POSIX threads

| Function/Type | Solaris Threads | POSIX Threads |
|-------------------------|---------------------------|-----------------------------------|
| header-file | <code>thread.h</code> | <code>pthread.h</code> |
| condition-variable type | <code>cond_t</code> | <code>pthread_cond_t</code> |
| lock a mutex | <code>mutex_lock()</code> | <code>pthread_mutex_lock()</code> |
| get a thread's ID | <code>thr_self()</code> | <code>pthread_self()</code> |

However, there are a number of areas where Solaris threads provides different functionality to POSIX threads. These include the following:

- daemon threads
- ability to suspend and continue a thread
- join-any-thread

To convert a Solaris threads application which uses these unique features to POSIX threads is likely to require a re-working of the application; the effort

involved may be considerable. This difficulty may be sufficient to discourage a software vendor from porting a Solaris application to Tru64 UNIX. In recognition of this, the SCL threads library provides Tru64 UNIX with most of the Solaris threads functionality.

The SCL threads library (STL) sits between the application and the underlying operating system's POSIX thread implementation (which on Tru64 UNIX is known as DECthreads). Thus performance of STL is not as optimal as using POSIX threads. However, the intention is to allow an application to be quickly ported.

3.2 STL Functionality

Table 3–2 shows the STL functionality supported for synchronization objects and thread functions.

Table 3–2: STL Functionality

| Synchronization Objects | Thread Functions |
|----------------------------------|---|
| mutexes ^{a b} | thread creation ^c |
| condition variables ^a | thread exit |
| read/write locks ^a | joining a thread ^d |
| semaphores | suspending and continuing a thread ^e |
| | thread-specific data |
| | thread concurrency |
| | thread priority ^f |
| | setting the signal mask of a thread |
| | getting the ID of a thread |
| | determining whether a thread is the main thread |
| | making a thread yield the CPU |

^a No system-wide (USYNC_PROCESS) support. Support is only process-wide (USYNC_THREAD).

^b No support for USYNC_PROCESS_ROBUST.

^c Includes detached, daemon and suspended thread creation, but not bound threads.

^d Includes *join-specific-thread* and *join-any-thread*.

^e Suspending and continuing a thread might, on rare occasions, fail. See Section 3.7.6.

^f API functions are provided to set and get a thread's priority, but these values do not affect the scheduling of the thread.

Appendix B lists all the functions provided. The reference pages document the restrictions of each function in more detail.

The following functionality is not supported:

- Support for mixing Solaris thread calls with POSIX thread calls
- Support for system-wide (USYNC_PROCESS) synchronization objects
- Support for USYNC_PROCESS_ROBUST mutexes
- Duplication of each thread when a `fork()` occurs.

3.3 Components of the Solaris Thread Library

The Solaris Thread Library component of the Solaris Compatibility Libraries comprises of two header files, `synch.h` and `thread.h`, and the shared object library `libthread.so`.

Do not confuse the Solaris Thread Library with the POSIX threads library at `/usr/shlib/libpthreads.so`.

There is no static version (`libthread.a`) of the STL. Any Tru64 UNIX system which is to run a Solaris thread program must have the Solaris Compatibility Libraries installed.

3.4 Building Applications Against STL

To build a Solaris thread application on Tru64 UNIX, the build-tools need to know where the Solaris thread files are. Assume that the STL library `libthread.so` resides in `/usr/opt/solcomplib/shlib` (the default location) and that the header-files `thread.h` and `synch.h` reside in `/usr/opt/solcomplib/include`. Then to compile, link and run Solaris thread programs on Tru64 UNIX, you need to do the following:

- To compile:

```
# cc -c -pthread -I/usr/opt/solcomplib/include prog.c
```

Note that you use the `-pthread` option in the `cc` command line to tell the compiler that your program is multi-threaded. This is equivalent to using the `-mt` compiler option on Solaris. The compiler option `-I/usr/opt/solcomplib/include` tells the compiler where to find the thread header files.

- To link:

```
# ld -call_shared prog.o -L/usr/opt/solcomplib/shlib -lthread -lc -o prog
```

Alternatively, you can do the linking via the `cc` command as follows:

```
# cc prog.o -L/usr/opt/solcomplib/shlib -lthread -o prog
```

Note that with either command, you use the link editor `-L` option to change the library directory search order for shared object libraries, so that `/usr/opt/solcomplib/shlib` is searched. Use the `-l` option to specify that the SCL thread library is to be used.

- Before attempting to run the program, ensure that the environment variables `LD_LIBRARY_PATH` and `NLSPATH` are correctly set, so that the loader can find the SCL libraries and message-catalog file respectively. For example, in the C shell, use commands like:

```
# setenv NLSPATH /usr/opt/solcomplib/lib/nls/msg/%N
# setenv LD_LIBRARY_PATH /usr/opt/solcomplib/shlib
# prog
```

3.5 STL Environmental Settings

A user can set a number of environment variables to override the default settings of the Solaris Thread Library. These settings include control of the following:

- The signals to be used for `thr_suspend()` and `thr_continue()`.
- The action to be taken if a Solaris function requests a feature that the Tru64 UNIX operating system doesn't support
- The attributes of newly-created daemon threads

The following list details the environment variables of special interest to STL users:

- `SCL_LOG_FILE`
 - The name of the log-file used by SCL for logging error messages.
 - Specify filename or `stdout / stderr`.
 - If `SCL_LOG_FILE` is not set, then no error messages are logged.

Note that `SCL_LOG_FILE` is generic to SCL, and not specific to STL.
See Chapter 5 for more information on error logging.
- `STL_SIG_SUSPEND`
 - An integer representing the signal number to use for `thr_suspend()`.
 - See `/usr/include/signal.h` for signal values.
 - If `STL_SIG_SUSPEND` is not set, then signal 30 (`SIGUSR1`) is used by default.
 - Ensure that `STL_SIG_SUSPEND != STL_SIG_CONTINUE`.
- `STL_SIG_CONTINUE`
 - An integer representing the signal number to use for `thr_continue()`.
 - See `/usr/include/signal.h` for signal values.

- If `STL_SIG_CONTINUE` is not set, then signal 31 (`SIGUSR2`) is used by default.
- Ensure that `STL_SIG_SUSPEND != STL_SIG_CONTINUE`.
- `STL_USYNC_PROCESS_ACTION`
 - A string representing the action to take when an attempt is made to create a system-wide synchronization object (such as a mutex) and the operating system version, or SCL, does not support it.
Tru64 UNIX Version 4.* does not support system-wide synchronization objects. Tru64 UNIX Version 5.0 does support system-wide synchronization objects, but this version of SCL does not support their use.
 - Possible values are:
 - `ABORT` -log message and abort process
 - `ERROR` -return error status from routine; don't log it
 - `IGNORE` -treat request for system-wide object as a process-wide object
 - If `STL_USYNC_PROCESS_ACTION` is not set, the default action is `ABORT`.
- `STL_DETACH_DAEMONS`
 - An integer. If 0, `STL_DETACH_DAEMONS` specifies that daemon threads should not be implicitly created as detached. Any other value causes daemon threads to be created as detached.
 - See Section 3.7.5 for information on joining threads and `STL_DETACH_DAEMONS`.
 - If `STL_DETACH_DAEMONS` is not set, then the default behaviour is to create daemon threads as detached.

If, for example, your Solaris thread application makes its own use of `SIGUSR1`, and the application uses `thr_suspend()`, then you must set `STL_SIG_SUSPEND` to another signal: one which is unused by the application.

Suggested alternative signals are:

```
24 : SIGXCPU
25 : SIGXFSZ
27 : SIGPROF
```

So depending upon your UNIX shell (this example assumes the C shell), you need to issue a command like the following before running your program:

```
# setenv STL_SIG_SUSPEND 24
```

3.6 Overview of the STL Implementation

The Solaris Thread Library for Tru64 UNIX is essentially a layer of software which sits between the application program and the operating system's underlying POSIX threads implementation.

3.6.1 Data types

Most of the Solaris data types are mapped onto the POSIX thread data-types. Although a type may be of different sizes on the two platforms, this should be mostly transparent to well-behaved applications that do not attempt to look inside these opaque data-types. The table below shows a couple of data-types for illustration.

Table 3–3: POSIX Threads Data Types

| Solaris Threads | Tru64 UNIX POSIX Threads |
|--|--|
| <code>mutex_t</code> : structure, size 24 bytes | <code>pthread_mutex_t</code> : structure, size 48 bytes |
| <code>thread_t</code> : 32-bit integer | <code>pthread_t</code> : 64-bit address |

Appendix A shows how STL maps the Solaris thread types onto the POSIX thread types on Tru64 UNIX.

3.6.2 Functions

With STL, most Solaris thread functions are implemented by calling the equivalent POSIX threads function. Often some conversion of the function arguments is needed.

In some cases, such as `thr_suspend()`, there is no direct POSIX thread equivalent to the Solaris thread function. Thus to implement the Solaris function, STL may have to make several run-time library, system and POSIX thread calls.

3.6.3 Performance

The primary goal of the Solaris Thread Library is to provide Solaris thread functionality for Tru64 UNIX. The STL implementation focuses on functionality, for ease of porting Solaris threaded code, rather than performance. There are a number of areas within STL where concurrency is restricted such that only one thread can be executing that routine at any time. These include the following:

- `thr_create()`: thread creation; and also thread termination
- `thr_suspend()`: suspending a thread

- `thr_continue()`: continuing a thread
- `thr_join()`: joining a thread

Solaris applications which attempt to have multiple threads executing one of these routines in parallel are likely to see poor performance.

For optimum performance on Tru64 UNIX, the Solaris thread application should be re-written to use POSIX threads, thus by-passing the Solaris Thread Library.

3.7 Issues When Running Solaris Thread Programs

This section presents issues that can arise when you run a Solaris thread program on Tru64 UNIX.

3.7.1 Displaying a Thread's Identifier

Solaris thread programs that display the thread-identifier type may need to be modified. On Solaris, the `thread_t` type is a 32-bit number, which starts at 1, and which can be displayed in 10 decimal digits. On Tru64 UNIX, the `pthread_t` type which `thread_t` is mapped onto is a 64-bit address, and would require 20 decimal digits to be displayed fully.

DECthreads on Tru64 UNIX does provide the function `pthread_getsequence_np()`, which returns an unsigned 64-bit integer sequence number for the thread. Typically the sequence numbers start from 1 for the first thread, and increase for subsequent threads. This is similar to the Solaris thread identifier, except that the thread sequence numbers are 64-bit values.

Note that with POSIX threads on Tru64 UNIX, thread identifiers can be re-used (as can thread sequence numbers). If a thread was created detached, then when the thread has completed its thread identifier can be re-allocated. When a non-detached thread completes, the thread identifier is not re-allocated until the thread has been joined.

One way an application could be changed to handle printing the different thread ID types is shown in the example code below.

```
#ifdef __osf__
/* Tru64 UNIX: thread ID is 64-bit address */
#define TID "0x%16p" /* or %20lu for decimal */
#endif

#ifdef __sun
/* Sun Solaris: thread ID is 32-bit integer */
#define TID "%10d"
#endif
```

```

status = thr_create( ..., &tid );
printf( "New thread ID is " TID "\n", tid );

```

3.7.2 Status Returns from Functions

Most Solaris thread and POSIX thread functions return an integer status value. If this status-return is 0, the function has succeeded. If the status-return is non-zero, then the function is indicating an error status. The Solaris thread APIs typically define a different set of error-status return values to POSIX threads. For example, consider the condition-wait functions:

Table 3–4: Return Status from condition-wait Functions

| Solaris Threads | POSIX Threads |
|-------------------------------------|--|
| cond_wait() Can return 0, EFAULT | pthread_cond_wait() Can return 0, EINVAL, or ENOMEM |

Ideally a threaded application checks that the return-status from the condition-wait function indicates success. The following code fragment gives an example.

```

status = cond_wait(...);
if (status != 0)
{
    /* we have an error */
    fprintf( stderr, "Error %d from cond_wait()\n", status );
    abort();
}

```

You can also perform the status-check by using `assert()`, as in the following example:

```

status = cond_wait(...);
assert( status == 0 );

```

But it's possible that Solaris thread application code actually checks for the error status rather than for success. For example,

```

status = cond_wait(...);
if (status == EFAULT)
{
    fprintf( stderr, "EFAULT error from cond_wait()\n" );
    abort();
}
else
{
    /* assume cond_wait() worked */
    ..
    ...
}

```

For this reason, STL maps function return values which indicate an error from the POSIX thread functions to one of the documented function return values which indicates an error for the Solaris thread function. Thus in the example of `cond_wait()`, if `pthread_cond_wait()` returns either `EINVAL` or `ENOMEM`, then STL maps these function return values from `pthread_cond_wait()` to `EFAULT` from `cond_wait()`.

When STL maps a function return value which indicates an error, it also logs a message which describes the mapping. For example, if `pthread_cond_wait()` returns `ENOMEM`, then a message like the following is logged to the file specified by `SCL_LOG_FILE`:

```
"Mapping error ENOMEM from system routine pthread_cond_wait()
to error EFAULT in Solaris routine cond_wait()"
```

Thus if you get an `EFAULT` status returned by STL's `cond_wait()` function, it does not necessarily have the same cause as it did on Solaris.

Note that a message is logged only if the function return value which indicates an error from the POSIX thread function differs to that documented for the Solaris thread functions. For example, the POSIX thread function `pthread_cond_timedwait()` is documented as returning `EINVAL` if any one of the function arguments is invalid, whereas Solaris threads documents that `EINVAL` from `cond_timedwait()` means that just the `abstime` parameter is invalid. Note also that Solaris threads and POSIX threads sometimes use different function return values which indicate an error for the same meaning. For example, `cond_timedwait()` returns `ETIME` if the function timed-out, whereas `pthread_cond_timedwait()` returns `ETIMEDOUT`. Because there is a mapping from POSIX thread function return value to the Solaris thread function return value, STL logs a message when `pthread_cond_timedwait()` returns `ETIMEDOUT`, and returns `ETIME`.

To summarise error-mapping:

- If the POSIX thread function returns an error status which matches that documented for the Solaris thread function, then the Solaris Thread Library does not log a message, even though the reason for the error status may be different.
- If the POSIX thread function returns an error status which differs to what is documented for the Solaris thread function, then the POSIX status is mapped to a Solaris status, and a message is logged indicating the mapping.

3.7.3 STL Aborts on Detecting an Internal Error

The Solaris Thread Library maintains various internal state lists. Various run-time library, POSIX thread and system routines are called internally to maintain this state. In the unlikely event that one of these routines returns

an unexpected error status, then STL logs a message and aborts the process, producing a core-dump. STL takes this action because an unexpected error means the STL internal state is potentially no longer valid, and to continue would only propagate the error and compromise the integrity of STL.

The most probable causes of such an error are software defects, such as over-writing memory by mistake, within the application or one of the libraries it uses (including the SCL libraries).

3.7.4 Incautious Use of `thr_suspend()` and `thr_continue()` Can Hang an Application

Solaris threads provides the functions `thr_suspend()` and `thr_continue()`, which are used to suspend a thread's execution, and to resume it, respectively. If a thread is suspended while it holds a mutex or read/write lock, then this will prevent other threads from locking the mutex. This can cause the application to appear to be hung.

Internally, STL uses mutexes and read/write locks to ensure its integrity is correctly maintained. STL takes action to prevent a thread from being suspended while it holds an internal-to-STL mutex or read/write lock. It does this by blocking the `suspend` signal, by calling `pthread_sigmask()`, before it takes out the lock. Once the thread is done with the lock, it unlocks it, and unblocks the `suspend` signal.

A Solaris application should take similar action to prevent the process from hanging, unless the application knows that when it suspends a thread, that the thread isn't holding any locks. Be aware that various run-time library routines, such as `printf()`, use their own locks, and need protecting too.

3.7.5 Joining threads

Solaris threads lets you join with any thread (that is, the first available joinable thread to terminate) if the first parameter to `thr_join()` is 0. POSIX threads only lets you join with a specific thread ID.

STL provides join-any-thread functionality, though a discrepancy between the Solaris documented behaviour and observed behaviour exists for `thr_join()` of daemon threads. The Solaris documentation states that daemon threads cannot be joined, but tests indicate that they can be joined if the daemon-thread ID is specified explicitly to `thr_join()`, and the daemon thread terminates. By default STL creates daemon threads as detached, meaning that they are not joinable. If you want daemon-threads to be created as joinable, then set the STL environment variable `STL_DETACH_DAEMONS` to 0 (see Section 3.5).

3.7.6 Suspending and Continuing a Thread Sometimes Fails

The `thr_suspend()` and `thr_continue()` functions, which suspend and resume a thread's execution, are implemented in STL by using signals. That is, one thread sends a signal to another thread to tell it to suspend or continue. Under rare circumstances, the Tru64 UNIX operating system might cause a signal to be lost. This can result in applications hanging when `thr_continue()` is called on a suspended thread. The problem is caused by the loss of the signal being sent by `thr_continue()` to the suspended thread. Because the STL implementation of `thr_continue()` waits for acknowledgement from the suspended thread that it has received the continue signal, the thread might hang inside `thr_continue()`.

This problem is known to affect Tru64 UNIX versions 4.0D, 4.0E, 4.0F and 5.0. No patch kits prior to 1-January-2000 contain a fix for this problem.

For the latest information on this issue, check the SCL Web page at <http://www.unix.digital.com/complibs>.

If your application does not use `thr_suspend()` and `thr_continue()`, then you will be unaffected by this problem.

Miscellaneous Components

This section documents the sets of library routines that fall into one of the following miscellaneous categories:

- signal names (Section 4.1)
- directory path (Section 4.2)
- high resolution timers (Section 4.3)
- asynchronous I/O (Section 4.4)
- large file support (Section 4.5)
- text domain (Section 4.6)
- wide character support (Section 4.7)
- run time dynamic linking (Section 4.8)
- DNS resolution (Section 4.9)
- remote user (Section 4.10)
- `tell()`, `snprintf()`, `vsnprintf()` (Section 4.11)
- `makedev()`, `major()`, `minor()` (Section 4.12)

4.1 Signal Name Library Routines

Solaris provides a number of library routines that deal with signal names and message text describing the signal. They are typically used within signal handlers to log messages regarding signals that have been processed.

The SCL provides implementations of the following library routines:

- `strsignal()` - get name of signal
- `psignal()` and `psiginfo()` - system signal messages
- `str2sig()` and `sig2str()` - translation between signal name and signal number

These library routines support only the set of signals that are supported on Tru64 UNIX, rather than Solaris, and therefore might not be entirely compatible with all applications. Similarly the text messages associated with a given signal might differ from those provided on Solaris. Consequently,

you might need to review use of these library routines and the signals that they deal with and make appropriate changes to source code.

See the SCL reference pages for additional information on using these library routines.

4.2 Directory Path Functions

The SCL provides an implementation of the `resolvepath()` system call, which resolves all symbolic links of a specified path name.

See the SCL reference pages for additional information on using this function.

4.3 High-resolution Timers

Solaris provides two library routines that can be used for application performance measurement:

- `gethrtime()` - returns the current high-resolution real time in nanoseconds
- `gethrvtime()` - returns the current high-resolution virtual time (CPU usage) in nanoseconds

As is the case with the Solaris high resolution timers, although the returned time value is in nanosecond units, the actual clock resolution is hardware dependent.

The `gethrvtime()` function returns the virtual time only when micro-state accounting is enabled by starting the application with the `/proc tool ptime` command.

The SCL provides an implementation of both library routines and the `ptime` command. Note that with the SCL, the `gethrtime()` function is sensitive to system time changes. If the clock is adjusted between two successive calls to this routine the time difference measured will not be the actual time - the amount of clock adjustment will be included. Therefore ensure that the system time is not changed when running applications that measure application performance with this function.

See the SCL reference pages for additional information on using these library routines.

The `ptime` utility is the only Solaris `/proc` tool provided in the SCL. The following are not available in the SCL or Tru64 UNIX:

- `pcrd`, `pfiles`, `pflags`, `pldd`, `pmap`, `prun`, `psig`, `pstack`, `pstop`, `ptree`, `pwait`, `pwdx`.

4.4 Asynchronous I/O Library Routines

Solaris provides four library routines that allow applications to perform asynchronous I/O to objects. These are provided in addition to the POSIX asynchronous I/O library routines.

The SCL implements all of the Solaris asynchronous I/O routines:

- `aioread()`, `aiowrite()` - initiate an asynchronous I/O operation
- `aiowait()` - wait for completion of asynchronous I/O operation
- `aio_cancel()` - cancel an asynchronous I/O operation

The implementation of the asynchronous read and write library routines on Tru64 UNIX uses a separate thread to call either `pread()` or `pwrite()`. Consequently, the behaviour of these library routines might differ to that on Solaris. In particular, some of the errors that could be returned by these functions before an I/O is issued on Solaris may get returned after the I/O is attempted on Tru64 UNIX.

In these cases, the return codes are placed in the result structure members `aio_return` and `aio_errno`, instead of being directly returned as a failed function call. To ensure correct error handling, you might need to change applications. Also, the return codes within the result structure members will reflect the possible return values for `pread()` and `pwrite()` on Tru64 UNIX.

Solaris applications that use these asynchronous I/O routines link against the `libaio.so` library. In Tru64 UNIX, `libaio.so` contains the POSIX asynchronous I/O routines. Users of the SCL-provided routines need to resolve the function symbols from `libsolmisc.so` or `libsolmisc.a`, rather than `libaio.so`.

The structure `aio_result_t` is defined differently by SCL. Although compatible with the Solaris definition, any source code that makes assumptions about the size or layout of this structure will need to change. See the SCL reference pages for additional information on using these library routines.

4.5 Large File Support

Solaris provides two variations of support for accessing large files (larger than 2GB). Firstly, there is the native "Largefile Compilation Environment" where standard file-related functions inherently operate on large files. This environment is a "feature" of the 64-bit application environment that can be used on Solaris. This environment is enabled by defining `_FILE_OFFSET_BITS=64` when compiling modules.

Tru64 UNIX is a 64-bit operating system and as such large file support is implemented as a standard feature. Therefore calls to standard functions do not need to change, and any use of `_FILE_OFFSET_BITS` is ignored by the SCL.

Secondly, Solaris has the "Transitional Compilation Environment" where 32-bit applications can access large files by calling 64-bit variants of standard file-related functions, for example, `open64()` and `fseek64()`. This environment is enabled by defining `_LARGEFILE64_SOURCE=1` when compiling modules.

Solaris applications may be explicitly calling 64-bit functions and defining and using 64-bit variants of standard types and structures in order to manipulate large files as well as normal (small) files. On Tru64 UNIX, the standard functions and types will be 64-bit capable and so the headers provided by SCL simply map uses of 64-bit variants to the standard Tru64 UNIX functions and types.

The SCL support for Solaris large files is within the C header files only. No functions are implemented.

4.6 Text Domain Library Routines

Solaris provides a number of library routines and commands that implement the UniForum `gettext()` message catalog system. This is in addition to the widely used X/Open Standard `catgets()` that is available in both Solaris and Tru64 UNIX. Tru64 UNIX does not provide the UniForum `gettext()` implementation by default, but a GNU version, the GNU `gettext` utilities, is available that builds and runs on Tru64 UNIX. This can be downloaded from <ftp://ftp.gnu.org/gnu/gettext/> or other GNU mirror sites. The `gettext` source package is also provided on the Freeware Source CD that ships with the Tru64 UNIX Operating System media.

Versions 0.10 and 0.10.35 of the GNU `gettext` utilities have been tested on Tru64 UNIX.

GNU `gettext` provides the same message handling library routines as Solaris:

- `gettext()`
- `dgettext()`
- `dcgettext()`
- `textdomain()`
- `bindtextdomain()`.

4.6.1 Documentation

You can view the `gettext` manual online and download it from <http://ftp.gnu.org/manual/gettext/index.html>.

Help on a command is available by entering the command name and then `-h`. For example, `xgettext -h`.

Solaris-equivalent `gettext` features are documented as reference pages that are supplied as part of the SCL.

Documentation about GNU `gettext` (and other GNU tools and libraries) is also supplied in GNU documentation format, known as Texinfo, rather than as a set of reference pages. Use the `info` command to invoke the documentation browser. You can download Texinfo from the GNU Internet site. The `info` binaries and support files are also available on the Freeware Executables CD that ships with the Tru64 UNIX Operating System media.

4.6.2 Installing GNU `gettext` Tool

The procedure that follows assumes you downloaded `gettext` Version 0.10.35 and that you have enabled root access.

To install the GNU `gettext` tools, do the following:

1. Move the downloaded file to the directory where you want to unpack the `gettext` files, and then `cd` to that directory. Unpack the downloaded file as follows (The `gunzip` utility ships with Tru64 UNIX):

```
# gunzip gettext-0.10.35.tar.gz
# tar xf gettext-0.10.35.tar
```

These commands create a `gettext-0.10.35` directory in the working directory, with a hierarchy of `gettext` files beneath it.

2. Read the file `gettext-0.10.35/INSTALL`, which gives generic instructions for installing GNU `gettext`. Note that the default location for installing files is directories beneath `/usr/local`. Accepting this default will mean environment differences to Solaris' commands and message file locations. Instructions on changing the default are given in `gettext-0.10.35/INSTALL`.
3. Execute the configure script. If `gcc`, the gnu C compiler is available on your system, use the command:

```
# ./configure
```

If `gcc` is not available, use the following command:

```
# CC=cc ./configure
```

4. Run the `make` command:

```
# make
```

5. Move the relevant files to their appropriate locations:

```
# make install
```

6. Update the relevant environment variables (for example `PATH`) to access the `gettext` commands and libraries. By default, the commands are located in `/usr/local/bin`, the static library is located in `/usr/local/lib`, and the header file is in `/usr/local/include`.

4.6.3 Using GNU `gettext` Tool

Use of GNU `gettext`, at both the command and API level, is the same as with Solaris, except GNU `gettext` provides a superset of the Solaris functionality.

Machine Object files (`.mo`) are generated from Portable Object files (`.po`) using the `msgfmt` command. To ensure UniForum behaviour as found in Solaris, use the `--strict` option to the `xgettext` command. The MO files need to be copied to the relevant locale directory that exists beneath `/usr/local/share/locale` by default.

Several locale-related directories are created when `gettext` is installed; for example `fr/LC_MESSAGES`, which are the target locations for `.mo` files.

The `gettext` command and API use the value of `LC_MESSAGES` environment variable to locate the locale directory. The current value can be viewed with the `locale` command. The value is changed by setting the `LANG` environment variable to one of the available locales, as output by the command `locale -a`. Typical locale names are `en_US.ISO8859-1`, or `fr_FR.ISO8859-1`. If you are going to use one of these locale names, you must create a directory with the name in `/usr/local/share/locale`.

Alternatively, if the simpler directory names are to be used (for example, `fr` or `de`), then you must set the `LC_MESSAGES` environment variable to that name. For example, `export LC_MESSAGES=fr`. Setting the `LANG` environment variable to this unsupported locale does not cause the appropriate changes to the category variables. Note that the value of the explicitly set `LC_MESSAGES` environment variable is not reflected in the output from the `locale` command.

4.7 Wide-character Support

Solaris provides a number of non-standard library routines that operate on Process Code (wide-character) strings. Many of these library routines have equivalents in the standard library, whereas a few are implemented using standard library routines.

The SCL provides support for the following wide-character library routines:

Wide-character String Operations, `wcstring()`:

`wscat()`, `wsncat()`, `wscmp()`, `wsncmp()`, `wscopy()`, `wsncpy()`, `wslen()`,
`wschr()`, `wsrchr()`, `wspbrk()`, `wsspnl()`, `wscspn()`, `wstok()`, `windex()`,
`wrindex()`, `wscoll()`, `wsxfrm()`.

Conversion from Wide Character to Long:

`watoi()`, `watol()`, `watoll()`, `wstol()`.

Conversion from Wide-character String to Double-precision:

`watof()`, `wstod()`.

Process Code string operations, `wstring()`:

`wscasecmp()`, `wsncasecmp()`, `wsdup()`, `wscol()`.

Formatted Input and Output Conversion:

`wsscanf()`¹, `wsprintf()`.

Code Conversion for Process Code and File Code:

`strtows()`, `wstostr()`.

Conversion between EUC characters and Process Code:

`getws()`, `putws()`.

Refer to the reference pages for these library routines for information on their use.

4.8 Runtime Dynamic Linking

Solaris and Tru64 UNIX both provide support for runtime dynamic linking with calls such as `dlopen()` and `dlsym()`. Solaris implements a number of library routines that are not provided with Tru64 UNIX:

`dladdr()`, `dlinfo()`, `dlDump()`, `dlmopen()`.

The SCL does not provide these library routines due to their complexity and Solaris-specific nature. Applications that use these library routines must use alternative mechanisms should the same functionality be required.

¹ Refer to `wsscanf(3scl)` for details on the SCL implementation of this function.

4.9 DNS Resolver Library Routines

Solaris and Tru64 UNIX both provide support for DNS resolver library routines that interact with BIND. Two of the functions, `res_search()` and `res_query()`, are not documented with Tru64 UNIX Version 4 although they are available with that version. Documentation of these is provided with Tru64 UNIX Version 5.0 and later.

BIND Version Incompatibilities

One issue that the SCL helps to overcome relates to differences in BIND versions. Solaris 2.* and Tru64 UNIX Version 4 both support BIND Version 4, whereas Solaris 7 and Tru64 UNIX Version 5.0 both support BIND Version 8. The `<arpa/nameser.h>` header is different for BIND Version 4 and BIND Version 8: they have different symbols that could appear in application code. Because of this, an application developed on Solaris 7 might use symbols that are not available if that application is ported to Tru64 UNIX Version 4. In this case, the source module will not build. The same problem would occur if you tried to compile the module in a Solaris 2.* environment.

The SCL provides a version of `<arpa/nameser.h>` that maps the new symbols from BIND Version 8 to existing symbols in BIND Version 4. This allows a BIND Version 8 application to build and execute in a Tru64 UNIX Version 4 environment as long as the application uses only BIND Version 4 features. Any new symbols that do not have old equivalents are not defined and consequently some applications might not build. It is assumed that use of such symbols indicates use of functionality not present in BIND Version 4. Such applications must be changed or built on Tru64 UNIX Version 5.

There is complete BIND version compatibility between Solaris 7 and Tru64 UNIX Version 5.0.

4.10 Remote User Library Routines

Solaris provides and documents two library routines that interact with the network username server `rpc.rusersd`. These library routines are `rusers()` and `rnusers()`. These library routines are provided with Tru64 UNIX, within `librpcsvc.a`, and therefore applications should be able to build successfully.

However, testing of these library routines on Tru64 UNIX V4 against Solaris 7 produced an error with `rnusers()` indicating a mismatch between the protocol versions in use.

4.11 Support for `tell()`, `snprintf()` and `vsnprintf()` Library Routines

The `tell()` library routine is provided and documented on Solaris. It is equivalent to calling `lseek(fd, 0, SEEK_CUR)`. The `tell()` function is provided on Tru64 UNIX but is undocumented. It performs the same task as on Solaris. An appropriate function prototype has been added to the SCL version of `<unistd.h>` in order to be compatible with Solaris.

The `snprintf()` and `vsnprintf()` library routines are provided and documented on Solaris and Tru64 UNIX Version 5.0. However, they are not present on Tru64 UNIX Version 4, and so an SCL-specific implementation of these library routines is provided, with the standard function names mapped to the SCL variants in the SCL version of `<stdio.h>`. This function mapping is unnecessary if building on Tru64 UNIX Version 5.0 or later, and consequently the header uses preprocessor statements to only map the definitions on Version 4.

To ensure that the correct preprocessor symbol is defined for the conditional test within the header, use the `tru64_version` utility (in the `bin` directory beneath the SCL root directory) with the application build command. For example:

```
# cc -o myprog -D`tru64_version` -m` myprog.c -lsolmisc
```

Alternatively, remove the definitions from the SCL-supplied `stdio.h` on a Version 5 system.

Note that on Solaris, the return values for these library routines represent the number of formatted characters generated by the library routines, which can be more than is output to the buffer. With Tru64 UNIX Version 5.0 the value returned reflects the number of characters actually stored in the buffer. Unlike Solaris and Tru64 UNIX, the SCL variants of these functions return an error if the output buffer is not large enough to contain the formatted characters.

4.12 Support for `makedev()` Function

The `makedev()`, `major()` and `minor()` library routines are provided as macros on Tru64 UNIX and defined in `<sys/types.h>`. You can get values for the major and minor device numbers with the command file `device_name`. For example,

```
# file /dev/disk/dsk10f
/dev/disk/dsk10f:      block special (19/203)
```

4.13 File-related Differences

There are a few differences between some of the file-related library routines and structures as implemented on Solaris and Tru64 UNIX. These differences are given in this section. You might need to change source code to adjust for some of these differences:

- The Solaris version of the `dirent` structure has different members than that of Tru64 UNIX. In particular, Solaris defines the `d_off` (offset) member, which is not defined on Tru64 UNIX. Applications referencing this member will need to change.
- The Solaris version of the `DIR` structure has different members than that of Tru64 UNIX. The Tru64 UNIX version is a superset; however, the member names conflict when determining whether POSIX source compliance is required. For example, Solaris member names are `d_xxx` for POSIX and `dd_xxx` for non-POSIX. Tru64 UNIX names the POSIX members `dd_xxx` or `__dd_xxx` depending on whether `OSF_SOURCE` is defined. If `OSF_SOURCE` is defined, then `__dd_xxx` symbols are mapped to `dd_xxx`.

To aid porting, the SCL `dirent.h` header maps the definitions of Solaris' `DIR` structure members to their equivalent in Tru64 UNIX. It is recognised that this can cause a conflict with other unrelated tokens (if used). In this case, you must make changes to the source code, or remove these definitions from the header by building with the preprocessor symbol `SCL_NO_MEMBER_DEFS` defined.

- Solaris allows the `O_LARGEFILE` flag to be specified with `open()`. This flag is not necessary on Tru64 UNIX. In Solaris source code, `O_LARGEFILE` might appear OR'd with other flags. To maintain source code compatibility this flag would ideally be defined by the SCL-supplied `<sys/fcntl.h>` header but there is no appropriate value to give this flag on Tru64 UNIX (that is, one that doesn't clash with others). Consequently this flag is defined as a descriptive string that makes clear that source code needs to change to eliminate use of the flag when compiled on Tru64 UNIX.
- On Solaris, the `flock` structure defines a member named `l_sysid`. This member is not implemented on Tru64 UNIX, therefore any source code references to it will need to be eliminated.
- Solaris supports the `F_FREESP` request with the `fcntl()` system call, but this isn't implemented with Tru64 UNIX. Consequently this flag is defined to be a descriptive string that makes clear that source code needs to change to eliminate use of the flag when compiled on Tru64 UNIX.
- On Solaris, the `getrlimit()` and `setrlimit()` system calls can, respectively, return or set values of `RLIM_SAVED_MAX` and

`RLIM_SAVED_CUR`. These values are not supported by the Tru64 UNIX implementations of the system calls. Instead, these flags are defined to be a descriptive string that makes clear that source code needs to change to eliminate use of the flag when compiled on Tru64 UNIX. These definitions are in the SCL-supplied `<sys/resource.h>` header file.

5

Error Logging

The Solaris Compatibility Libraries for Tru64 UNIX will, as much as possible, return the same error codes that are returned on Solaris, even though there may be different errors occurring within the implementation. In this case the actual error is mapped to the "best-fit" error that the Solaris function can return.

In order to aid debugging, when an error within the implementation occurs and is mapped to a different error code, SCL will write an entry to the SCL error logfile that indicates which actual error occurred. This error log can thus be used to aid application debugging or when reporting problems with SCL.

Use of the SCL error log is enabled by defining the `SCL_LOG_FILE` environment variable to be the name of the file to which errors are written, or `stdout` or `stderr`. No messages will be output if `SCL_LOG_FILE` is undefined.

The content of the error log is similar to the following:

```
0.00008|0001: SCL Logfile initialized
0.00016|0001: Failed to initialize aio routines
0.00016|0001: SCL Logfile being closed
```

The numbers prefixing the message represent the time since the log file was initialized (in seconds) and the thread sequence number that is logging the message.

The messages written to the log file are obtained from a message catalog named `scl_message.cat`. This catalog resides in the `lib/nls/msg` directory beneath the SCL root directory. With this version of SCL there is no support for internationalized messages.

In order to locate the message file, one of two mechanisms can be used:

- If the `LANG` environment is defined when the install script is executed, it will create a link in `/usr/lib/nls/msg/$LANG` to the catalog residing beneath the SCL root directory. This directory is searched by default.
- If `LANG` is not defined, or is different when applications using SCL execute, then `NLSPATH` environment variable must be used to specify the location of the catalog. Define or modify `NLSPATH` to reference the

message directory beneath the SCL root directory. This can be achieved with a command similar to the following:

```
# export NLSPATH=$NLSPATH:/usr/opt/solcomplib/lib/nls/msg/%N
```

Restrictions

If a process that has enabled use of SCL error log forks then error logging is consequently disabled for the new process. It remains enabled for the existing process.

6

Reference Pages

Version 1.1 of the SCL provides reference pages for all implemented functions and commands. These reference pages reside in the `man` directory hierarchy beneath the SCL root directory. For the `man` command to search the SCL `man` directory, you need to define or modify the `MANPATH` environment variable. In C shell, you can do this as follows:

```
# setenv MANPATH /usr/opt/solcomplib/man:$MANPATH
```

The reference pages for SCL are within the usual numeric sections and are suffixed with `scl`. For example, `scl_intro(3scl)` and `ptime(1scl)`. You can use this suffix in the `man` command to specify SCL reference pages, if it is necessary to distinguish SCL functions and commands from other pages. For example:

```
# man 3scl aioread
```

The reference pages for SCL RPC commands and functions refer the reader to the reference pages that are supplied with the public-domain version of TI-RPC on which the SCL RPC is based. These pages are also supplied and reside in the `man/sun` directory beneath the SCL root directory. To reference the Sun pages it is necessary to additionally specify the `man/sun` directory with `MANPATH`.

The SCL reference pages are also supplied in HTML format in the `doc/ref/html` directory beneath the SCL root directory. Use `index.html` to link to all other reference pages.

Building the SCL Source Code

This chapter explains how to unpack and install the SCL source code. If you do not intend to use the source code, skip the chapter.

The source code for SCL is publicly available by downloading from the SCL web site or by installing the `SCLSOURCE nnn` subset. Either of these will give a `tar` file containing the SCL development source tree. Installation of the subset places the `tar` file in the `source` directory beneath the SCL root directory. The SCL source code has not been built, installed, or tested on versions of Tru64 UNIX prior to Version 4.0D.

7.1 Unpacking the Source Files

Before building the SCL source code, you need to create the source tree by unpacking the `tar` file. If you intend to modify the source, it is recommended that you create the source tree in a location that is distinct from the SCL root directory. This eliminates potential confusion that may arise if changed source modules and build output files are located beneath the SCL root created when SCL was originally installed.

The source tree can reside at any location, and may typically be beneath a developer's home directory. The following commands can be used to create the tree in `directory/my_scl_dir` and unpack the `tar` file there. They assume that the SCL source `tar` file, `sclsource.tar`, is located in the `path` directory.

```
# mkdir directory/my_scl_dir
# cd directory/my_scl_dir
# tar -xf path/sclsource.tar .
```

Figure 7-1 shows the top three branches of the resultant source tree hierarchy. This tree along with the sub-directories not depicted is collectively referred to as the SCL source directories.

Different versions of SCL may be in place on a system. The original distribution is in `/usr/opt/SCLnnn`, but other private, modified versions may exist elsewhere. It is recommended that you maintain a link to the active or preferred version, so that application developers can always refer to the latest version when building their applications. The suggested link is `/usr/opt/solcomplib`, which ordinarily links to directory `/usr/opt/SCLnnn`.

When a new SCL run-time tree is ready for deployment, copy it to a system directory, for example `/usr/opt/my_scl_dir`. Modify the `solcomplib` link to refer to it. Typically, users will have `/usr/opt/solcomplib` in their `PATH` and `LD_LIBRARY_PATH` environment variables. If this isn't the case, they must change these environment variables to reference the new directory.

7.5 Redistributing the Run-Time and Source Directories

If you modify the SCL source tree and use the resultant run-time with applications, then you need to redistribute the SCL run-time directories to systems that will run the applications.

No mechanism is supplied for packaging the necessary files into an installable kit. It is suggested that you pack the required SCL run-time directories into a `tar` file and unpack them on the target systems in an appropriate system directory.

The SCL run-time directories are shown in Figure 7-2. These include documentation and examples which are not required for a run-time only distribution. The minimal directories to package for runtime-only support are:

- `bin`
- `etc`
- `lib`
- `shlib`
- `include`

Package the `include` directory if it is likely that applications will be built using SCL on the target system.

- The `COPYRIGHT-DISCLAIMER` notice must be included in any distribution package.

It may also be appropriate to package and distribute the modified source tree. Include all the directories and files shown in Figure 7-1 (including the sub-directories not depicted) in this source package. You do not need to

include any top-level directories not shown in Figure 7-1, unless source tree modifications have created additional ones.

A

Mapping Solaris thread types to POSIX thread types by STL

From <synch.h> and <thread.h>:

```
struct timestruct {
    long    tv_sec;        /* seconds */
    long    tv_nsec;      /* nanoseconds */
};

typedef struct timestruct timestruc_t; /* used by cond_timedwait() */

#define cond_t          pthread_cond_t
#define mutex_t         pthread_mutex_t
#define rwlock_t       tis_rwlock_t
#define sema_t          sem_t

typedef pthread_t      thread_t;
typedef pthread_key_t  thread_key_t;

#define DEFAULTMUTEX   PTHREAD_MUTEX_INITIALIZER
#define DEFAULTTCV     PTHREAD_COND_INITIALIZER
#define DEFAULTRWLOCK TIS_RWLOCK_INITIALIZER

Solaris Type                      Tru64 UNIX v4.0F
-----                          -
cond_t                             pthread_cond_t
<synch.h>                          <thread.h>
structure; 16 bytes                 structure; 40 bytes

mutex_t                             pthread_mutex_t
<synch.h>                          <thread.h>
structure; 24 bytes                 structure; 40 bytes

rwlock_t                           tis_rwlock_t
<synch.h>                          <thread.h>
structure; 64 bytes                 structure; 152 bytes

sema_t                              sem_t
<synch.h>                          <semaphore.h>
structure; 48 bytes                 structure; 4 bytes
typedef psx4_key_t sem_t;
<sys/psx4_nspace_ts.h>
(indexes into semaphore array)

thread_key_t                        pthread_key_t
<thread.h> => <sys/types.h>
unsigned int; 4 bytes               unsigned int; 4 bytes
```

```
thread_t
  <thread.h>
  unsigned int; 4 bytes
```

```
pthread_t
  <thread.h> => <sys/types.h>
  address (pointer); 8 bytes
```

B

Solaris thread functions implemented by STL

Routines are ordered alphabetically.

```
cond_broadcast()
cond_destroy()
cond_init()
cond_signal()
cond_timedwait()
cond_wait()

mutex_destroy()
mutex_init()
mutex_lock()
mutex_trylock()
mutex_unlock()

rwlock_destroy()
rwlock_init()
rw_rdlock()
rw_tryrdlock()
rw_trywrlock()
rw_unlock()
rw_wrlock()

sema_destroy()
sema_init()
sema_post()
sema_trywait()
sema_wait()

thr_continue()
thr_create()
thr_exit()
thr_getconcurrency()
thr_getprio()
thr_getspecific()
thr_join()
thr_keycreate()
thr_kill()
thr_main()
thr_min_stack()
thr_self()
thr_setconcurrency()
thr_setprio()
thr_setspecific()
thr_sigsetmask()
thr_stksegment()
thr_suspend()
thr_yield()
```


C

SCL Installation Example

The following is an example of installing the SCL subsets:

```
# setld -l .
```

The subsets listed below are optional:

There may be more optional subsets than can be presented on a single screen. If this is the case, you can choose subsets screen by screen or all at once on the last screen. All of the choices you make will be collected for your confirmation before any subsets are installed.

- 1) Solaris Compatibility Libraries (SCL) Commands, Runtime and Header Files
- 2) Solaris Compatibility Libraries (SCL) Documentation and Man Pages
- 3) Solaris Compatibility Libraries (SCL) Source Code

Or you may choose one of the following options:

- 4) ALL of the above
- 5) CANCEL selections and redisplay menus
- 6) EXIT without installing any subsets

Estimated free disk space(MB) in root:44.6 usr:551.7

Enter your choices or press RETURN to redisplay menus.

Choices (for example, 1 2 4-6): 4

You are installing the following optional subsets:

Solaris Compatibility Libraries (SCL) Commands, Runtime and Header Files
Solaris Compatibility Libraries (SCL) Documentation and Man Pages
Solaris Compatibility Libraries (SCL) Source Code

Estimated free disk space(MB) in root:44.6 usr:542.7

Is this correct? (y/n): y

Checking file system space required to install selected subsets:

File system space checked OK.

3 subset(s) will be installed.

Loading subset 1 of 3 ...

Solaris Compatibility Libraries (SCL) Commands, Runtime and Header Files
Copying from . (disk)
Verifying

Loading subset 2 of 3 ...

Solaris Compatibility Libraries (SCL) Documentation and Man Pages
Copying from . (disk)
Verifying

Loading subset 3 of 3 ...

Solaris Compatibility Libraries (SCL) Source Code
Copying from . (disk)
Verifying

3 of 3 subset(s) installed successfully.

Configuring "Solaris Compatibility Libraries (SCL) Commands, Runtime and Header Files" (SCLBASE110)

SCLBASE110 has been loaded into /usr/opt/SCL110.

You may want to create the symbolic link from
/usr/opt/solcomplib to /usr/opt/SCL110
E.g.

```
ln -s /usr/opt/SCL110 /usr/opt/solcomplib
```

Create symbolic link now ([y]/n)?: y
Symbolic link /usr/opt/solcomplib created.

SCLBASE110 has a post-installation command-script at
/usr/opt/SCL110/install which completes the setup
of SCLBASE110.

Run the install script now ([y]/n)?: y
>>>>>>>

Performing SCL installation tasks...

LANG is not defined. You should ensure NLSPATH references
the SCL message catalog in /usr/opt/solcomplib/nls/msg

SCL installation tasks completed
<<<<<<<

Configuring "Solaris Compatibility Libraries (SCL) Documentation and Man Pages" (SCLDOC110)

SCLDOC110 has been loaded into /usr/opt/SCL110.

The SCL User's Guide is installed in the /usr/opt/SCL110/doc
directory in PDF, Postscript and text formats.

The SCL User's Guide is also supplied in HTML format. Use your
browser to open /usr/opt/SCL110/doc/html/scl_ug.html

The SCL110 man pages are installed under the
/usr/opt/SCL110/man directory. You will need to alter the
MANPATH environment variable to refer to these man pages.

For example, if the symbolic link /usr/opt/solcomplib has been
created, you should issue a command like:

```
csh> setenv MANPATH /usr/opt/solcomplib/man:$MANPATH
```

The SCL man pages have the "scl" suffix, for example,
scl_intro(3scl).

Configuring "Solaris Compatibility Libraries (SCL) Source Code" (SCLSOURCE110)

SCLSOURCE110 has been loaded into /usr/opt/SCL110.

The source code is located in the tar-file sclsources.tar
in the directory /usr/opt/SCL110/source.

Refer to the SCL User's Guide for information on unpacking and
building the SCL source code.

Index

Numbers and Special Characters

64-bit pointers, 1-1

A

aiocancel(), aioread(), and aiowait(),
4-3
asynchronous I/O library routines,
4-3

B

big-endian issues, 1-1
building applications, 1-6
 SCL RPC libraries, and, 2-6
 threaded applications, 3-3
building the SCL components, 7-2

C

cc -L, 1-6
command search path, 1-7
compiling applications
 threaded applications, 3-3
configuring XTI kernel component,
2-5

D

development tools, 1-2
differences between Solaris and SCL
 asynchronous library routines, 4-3
 /proc utility, 4-2

directory path functions, 4-2
directory search order
 shared object libraries, 3-3
directory structure of the SCL, 1-4
distributing the run-time and source
 directories, 7-4
documentation, 6-1
 Accessing Sun books online, 1-2
 Accessing Tru64 UNIX books
 online, 1-2

E

endian issues, 1-1
environment variables
 LD_LIBRARY_PATH, SCL library
 path, 2-6, 3-3
 MANPATH, reference page search
 path, 6-1
 NLSPATH, message-catalog file,
 3-3
 PATH, command search path, 1-7
 SCL_LOG_FILE, message logging,
 2-8
 SCL_ROOT_DIR, SCL root
 directory, 1-3, 7-3
/etc/netconfig, 2-5
/etc/services
 scl_rpcbind service, 2-3
examples
 square example, 2-7

F

file locations, 1-4

G

gethrtime() and gethrtime() library routines, 4-2

H

header files
installing, 1-5
high resolution timers, 4-2

I

include files
installing, 1-5
installation, 1-4
SCL source code, 7-1
setld example, C-1
Internet services
scl_rpcbind, 2-3

K

kernel components
configuring XTI, 2-5
known problems
RPC, 2-8

L

LD_LIBRARY_PATH environment variable, 2-6, 3-3
libraries, 1-6
linking with libxti, 2-7
specifying SCL libraries, 1-6
library path
setting, 2-6
setting the SCL library path, 3-3
libthread.so, 3-3
libxti, 2-7
linking
libxti, 2-7
linking libraries
linking with SCL libraries, 1-6

little-endian issues, 1-1
logging
message logging file, 2-8
lsolmisc SCL miscellaneous functions library, 1-6
lthread SCL threads library, 1-6

M

man pages, 6-1
search path, 6-1
MANPATH reference page search path, 6-1
message logging file, 2-8
message-catalog file
setting the search path, 3-3
mutexes
restrictions on, 3-3

N

NLSPATH environment variable, 3-3

P

port number, 2-2
Porting Assistant, 1-1
portmap
Sockets RPC port number, 2-2
POSIX threads
converting from Solaris to POSIX threads, 3-1
/proc utility, 4-2
process control, 4-2
ptime command, 4-2

R

redistributing the run-time and source directories, 7-4
reference pages, 6-1
search path, 6-1
required software, 1-2

- resolvepath() system call, 4-2
- restrictions
 - RPC, 2-8
 - threads functionality, 3-3
- root directory
 - referencing the SCL root directory, 1-3, 7-3
- RPC
 - building applications, 2-6
 - checking the SCL RPC
 - configuration, 2-4
 - known problems, 2-8
 - netconfig file, 2-5
 - restrictions, 2-8
 - SCL RPC, 2-2
 - secure RPC not supported, 2-8
 - setting up, 2-4
 - Sockets RPC port number, 2-2
- rpcbind command, 2-2, 2-4
- rpcgen command, 2-3
- rpcinfo command, 2-4
- run-time directories
 - location of, 7-2
 - using, 7-3
- run-time directory
 - distributing, 7-4

S

- SCL directory structure, 1-4
- SCL libraries
 - building applications with, 1-6
- SCL root directory
 - referencing the SCL root directory, 1-3, 7-3
- SCL source code
 - installing, 7-1
- SCL threads library
 - (*See* STL)
- SCL_LOG_FILE environment
 - variable, 2-8
- scl_netconfig file, 2-4, 2-5

- SCL_ROOT_DIR environment
 - variable, 1-3, 7-3
- scl_rpcbind service, 2-3
- SCLBASE, SCLBASE, and SCLSOURCE software subsets, 1-4
- sclbuild script, 7-2
- secure RPC not supported, 2-8
- setld
 - example, C-1
 - installing the SCL, 1-4
- setting up RPC, 2-4
- shared libraries, 1-6
- shared object libraries
 - directory search order, 3-3
- shlib, 1-6
- signal name library routines, 4-1
- Sockets RPC
 - port number, 2-2
- software development tools, 1-2
- software subsets, 1-4
- Solaris porting guide, 1-2
- source code
 - installing the SCL source code, 7-1
- source directory
 - distributing, 7-4
- source tree
 - building, 7-2
 - location of, 7-1
- square example, 2-7
- STL functionality, 3-2
- structure of the SCL, 1-4
- Sun online documentation, 1-2
- synch.h, 3-3
- synchronization objects
 - restrictions on, 3-3

T

- TCP port number, 2-2
- test_sclrpc_config, 2-4
- testing the run-time directories, 7-3

thread.h, 3-3
threads
 building applications, 3-3
 converting from Solaris to POSIX
 threads, 3-1
 libraries, 3-3
 STL functionality, 3-2
 unsupported functionality, 3-3
timers
 high resolution timing, 4-2
tree structure of the SCL, 1-4
Tru64 UNIX online documentation,
 1-2

U

UDP port number, 2-2
unsupported

 /proc commands not supported, 4-2
unsupported threads functionality,
 3-3
/usr/opt/solcomplib, 1-4
/usr/opt/solcomplib/etc/scl_netconfig,
 2-5
/usr/opt/solcomplib/shlib, 1-6

X

X/Open Transport Interface
 (*See* XTI)
XDR
 no support for long doubles, 2-8
XTI
 configuring, 2-5
 linking with libxti, 2-7
xtiso, 2-5

How to Order Tru64 UNIX Documentation

You can order documentation for the Tru64 UNIX operating system and related products at the following Web site:

<http://www.businesslink.digital.com/>

If you need help deciding which documentation best meets your needs, see the Tru64 UNIX *Documentation Overview* or call **800-282-6672** in the United States and Canada. In Puerto Rico, call **787-781-0505**. In other countries, contact your local Compaq subsidiary.

If you have access to Compaq's intranet, you can place an order at the following Web site:

<http://asmorder.nqo.dec.com/>

The following table provides the order numbers for the Tru64 UNIX operating system documentation kits. For additional information about ordering this and related documentation, see the *Documentation Overview* or contact Compaq.

| Name | Order Number |
|---|---------------------|
| Tru64 UNIX Documentation CD-ROM | QA-6ADAA-G8 |
| Tru64 UNIX Documentation Kit | QA-6ADAA-GZ |
| End User Documentation Kit | QA-6ADAB-GZ |
| Startup Documentation Kit | QA-6ADAC-GZ |
| General User Documentation Kit | QA-6ADAD-GZ |
| System and Network Management Documentation Kit | QA-6ADAE-GZ |
| Developer's Documentation Kit | QA-6ADAG-GZ |
| Reference Pages Documentation Kit | QA-6ADAF-GZ |

Reader's Comments

Solaris Compatibility Libraries for Compaq Tru64 UNIX User's Guide

Compaq welcomes your comments and suggestions on this manual. Your input will help us to write documentation that meets your needs. Please send your suggestions using one of the following methods:

- This postage-paid form
- Internet electronic mail: `readers_comment@zk3.dec.com`
- Fax: (603) 884-0120, Attn: UBPG Publications, ZKO3-3/Y32

If you are not using this form, please be sure you include the name of the document, the page number, and the product name and version.

Please rate this manual:

| | Excellent | Good | Fair | Poor |
|---|--------------------------|--------------------------|--------------------------|--------------------------|
| Accuracy (software works as manual says) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Clarity (easy to understand) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Organization (structure of subject matter) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Figures (useful) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Examples (useful) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Index (ability to find topic) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Usability (ability to access information quickly) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

Please list errors you have found in this manual:

| Page | Description |
|-------|-------------|
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |

Additional comments or suggestions to improve this manual:

What version of the software described by this manual are you using? _____

Name, title, department _____

Mailing address _____

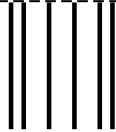
Electronic mail _____

Telephone _____

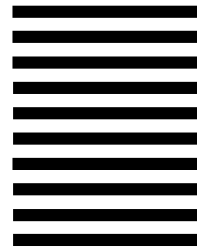
Date _____

----- Do Not Cut or Tear - Fold Here and Tape -----

COMPAQ



NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL
FIRST CLASS MAIL PERMIT NO. 33 MAYNARD MA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPAQ COMPUTER CORPORATION
UBPG PUBLICATIONS MANAGER
ZKO3-3/Y32
110 SPIT BROOK RD
NASHUA NH 03062-2698



----- Do Not Cut or Tear - Fold Here -----

Cut on This Line