



OSP Toolkit

Porting Guide

| Release 3.1.2

| 1 July 2004

Revision History

Revision	Date of Issue	Changes
2.7.0	March 12 th , 2003	Added revision history.
2.8.0	March 20 th , 2003	No Changes
2.8.1	March 25 th , 2003	No Changes
2.8.2	April 9 th , 2003	No Changes
2.9.0	June 1 st , 2003	No Changes
2.9.1	July 7 th , 2003	No Changes
2.9.2	July 28 th , 2003	Added documentation for Memory related performance enhancements. Modified the Introduction to include the Interoperability documents.
2.9.3	Sep 15 th , 2003	No changes
2.11.1	Feb 12 th , 2004	Removed comments on non-secure mode of compilation.
3.0	March 11 th , 2004	No changes
3.1	April 8 th , 2004	No changes
3.1.1	May 12 th , 2004	Changes for mtmalloc.
3.1.2	July 1 st , 2004	Updated the compilation section to point to the 'How to Build the toolkit' document.

Contents

Revision History	2
Contents	3
Introduction	4
Getting Started	4
Operating System	5
Replaceable Components	5
Required Components	6
Source Roadmap	6
Toolkit Naming Conventions	6
File Layout	6
Header files	6
Source files	8
Miscellaneous files	10
Building the OSP Toolkit	10
Supported compilers	10
Compiling under Windows NT/Windows2000	10
Compiling under Solaris	Error! Bookmark not defined.
Frequently Asked Questions	10
What Versions of OSP does the Toolkit Support?	10
Why does the Toolkit spell Authorisation with an "s"?	10
Is the Toolkit thread-safe?	11
Does the Toolkit support non-blocking calls?	11
What is the default memory management scheme implemented in the toolkit and how does it affect the toolkit performance?	11

E-mail: support@transnexus.com

www.transnexus.com

Copyright © 2003 by TransNexus. All Rights Reserved.

Introduction

This document provides a porting guide for of release 3.1.2 of the Open Settlement Protocol (OSP) Toolkit. That Toolkit, available freely under license from TransNexus, contains an implementation of the standard settlement protocol endorsed by the European Telecommunications Standards Institute (ETSI) and the International Multimedia Teleconferencing Consortium's Voice over IP (VoIP) Forum. The Toolkit also implements, as an option, extensions to the standard that allow access to enhanced services.

The OSP Toolkit contains fourteen separate documents, including this one. The documents are:

- *Introduction*
- *H.323 Implementation Guide*
- *SIP Implementation Guide*
- *How to Build and Test the OSP Toolkit*
- *Error code List*
- *Programming Interface*
- *Cisco Interoperability Example*
- *Device Enrollment*
- *Internal Architecture*
- *Porting Guide*
- *SIP – OSP Interoperability Test Cases*
- *H.323 – OSP Interoperability Test Cases*
- *Protocol Extensions*
- *ETSI Technical Specification TS 101 321*

The *OSP Toolkit Introduction* includes a “Document Roadmap” section that summarizes the various documents and their application.

The sections that follow provide a high-level overview of the porting process, a roadmap for the source code, directions for building the distribution, and instructions for testing an implementation using the Toolkit's test tools. This document concludes with a list of frequently asked questions, with answers.

Getting Started

The OSP Toolkit is designed for easy porting to a wide variety of operating environments. It is available as source code complying with ANSI C standards and, subject to intellectual property constraints, includes all software required for a fully functional OSP client. The software is also isolated from platform-specific details through a small collection of macros, and it is modularized to allow the replacement of Toolkit components with other tools that may already be available on a target platform.

When beginning a port to a new platform, developers should understand the operating system requirements of the Toolkit, and decide which Toolkit components to replace with tools already present in the target environment. Finally, developers should decide on a strategy for providing the additional required components. This section provides an overview of these issues.

Operating System

TransNexus has developed and fully tested the OSP Toolkit in both Solaris 2.7, Solaris 2.8, Linux 8.0, Windows2000 and Windows NT 4.0 environments. With appropriate configuration settings, the Toolkit source will compile *as is* in either environment. No porting is necessary. Other platforms substantially similar to Windows NT 4.0 or Solaris 2.7/2.8, including Windows 2000 and other UNIX systems, may also support the Toolkit without porting; however, TransNexus has not exhaustively tested the Toolkit in those environments.

The Toolkit requires a minimum number of services from any target environment. Required services are

- ANSI-standard memory management,
- POSIX Thread support, and
- Socket-based network input/output

The Toolkit isolates access to these services by providing a small set of macros. Any changes made to accommodate a new target environment should be made to the macros, rather than the Toolkit source itself.

Replaceable Components

The Toolkit provides a complete implementation of an OSP client. The Open Settlement Protocol relies on many common network technologies, and the Toolkit includes minimal support for all of those technologies, except those subject to intellectual property constraints. Since these technologies are common, however, many target environments may already include their own support. The Toolkit source is modularized to facilitate replacing its own components with support already existing in the target environment. Replaceable components include

- Abstract Syntax Notation One (ASN.1) processing
- Base64 encoding and decoding
- Extensible Markup Language (XML) processing
- Hypertext Transfer Protocol (HTTP) processing
- Multipurpose Internet Mail Extension (MIME) processing
- Public Key Cryptography Standards (PKCS) processing
- Secure MIME (S/MIME) processing
- X.509 certificate processing

Note that, in general, the Toolkit components provide only the minimal support necessary for OSP and are not suitable for general-purpose use.

Required Components

Because of intellectual property constraints, two components essential to OSP operation are not included with the Toolkit. Those components are cryptographic algorithms and secure socket layer (SSL) processing. As delivered, the Toolkit assumes the availability of the OpenSSL SSL implementation, and optionally the BSAFE cryptographic library.

Note that the Toolkit source code may include configuration options for support of other cryptographic or SSL packages. If appropriate to a target environment, these options may be used as a starting point for the port; however, the presence of these configuration options does not imply that TransNexus has thoroughly tested the Toolkit with the indicated component.

Source Roadmap

Because the software development kit is delivered as C-language source code, an overall understanding of the organization and structure of that source code is helpful in any porting effort. This section explains the Toolkit's naming conventions, and it outlines the contents of the files included in the kit. To minimize conflict and ensure that the Toolkit is consistently documented, this section only includes a brief overview of the source files. More detailed documentation may be found in the source code itself.

Toolkit Naming Conventions

To aid in recognizing the source code, and to avoid name space pollution, the Toolkit uses a naming convention for all objects it contains. The following table summarizes those conventions.

Prefix	Object Type
OSPC_	constant
OSPC_ERR	error code
OSPM_	(parameterized) macro
OSPP	global function (procedure)
osp	local function (procedure)
OSPS	structure
OSPT	type definition
OSPV	global variable

File Layout

The following subsections list the files included in the OSP Toolkit, along with a brief description of each file. The Toolkit contains C header (.h) files, C source (.c) files, and a few miscellaneous support files.

Header files

OSP Toolkit Header Files	
osp.h	Public application programming interface and constants

OSP Toolkit Header Files	
ospaltinfo.h	Alternate information element
ospasn1.h	ASN.1 processing
ospasn1ids.h	ASN.1 object identifiers
ospaudit.h	Auditing (message signing) functionality
ospauthcnf.h	AuthorisationConfirmation message element
ospauthind.h	AuthorisationIndication message element
ospauthreq.h	AuthorisationRequest message element
ospauthrsp.h	AuthorisationResponse message element
ospb64.h	Base64 encoding and decoding
ospbfr.h	Message buffers
ospbsafetstd.h	BSAFE interface
ospcallid.h	Call identifier tracking
ospciscoext.h	Cisco-specific extensions
ospcode.h	Status codes
ospcomm.h	Communication manager object
ospconfig.h	Configuration parameters
ospcrypto.h	Cryptographic processing
ospcryptowrap.h	Wrapper functions for cryptographic library interfaces
ospcustomdebug.h	Debugging and logging customization
ospdatatypes.h	Miscellaneous data types
ospdebug.h	Debugging and logging
ospdest.h	Destination message element
osperrno.h	Error numbers
ospfail.h	Setup failure reasons
osphttp.h	HTTP processing
ospinit.h	Initialization
ospkeys.h	Cryptographic keys
osplist.h	Linked lists
ospmime.h	MIME processing
ospmsg.h	General message object
ospmsgattr.h	Message attributes
ospmsgdesc.h	Message descriptors
ospmsgelem.h	Message elements
ospmsginfo.h	Message information object
ospmsgpart.h	Message parts
ospmsgque.h	Message queues
osposincl.h	Common operating system interface
ospossys.h	Operating system dependencies
ospostime.h	Time-of-day
osppkcs1.h	Public key cryptography standard # 1
osppkcs7.h	Public key cryptography standard # 7
osppkcs8.h	Public key cryptography standard # 8
ospprovider.h	Provider object
ospproviderapi.h	Provider interface
ospreauthreq.h	ReauthorizationRequest message element
ospreauthrsp.h	ReauthorizationResponse message element
ospsecurity.h	Security object
ospsocket.h	Socket interface

OSP Toolkit Header Files	
ospssl.h	Common SSL processing
ospsslsess.h	SSL session object
ospstatistics.h	Statistics processing
ospstatus.h	Status message element
osptnaudit.h	transnexus.com: Audit message element
osptnlog.h	Logging functionality
osptnprobe.h	QoS probing functionality
osptoken.h	Token object
osptokeninfo.h	Token information
osptrans.h	Transaction object
osptransapi.h	Transaction interface
osptransid.h	TransactionId message element
ospusage.h	Usage object
ospusagecnf.h	UsageConfirmation message element
ospusageind.h	UsageIndication message element
osputils.h	Utility functions
ospx500.h	X.500 objects
ospx509.h	X.509 certificate objects
ospxml.h	XML processing
ospxmlattr.h	XML attribute processing
ospxmldoc.h	XML document object
ospxmlem.h	XML element object
ospxmltype.h	XML character type processing

Source files

OSP Toolkit Source Files	
ospaltinfo.c	Alternative information objects
ospasn1.c	ASN.1 processing
ospasn1ids.c	ASN.1 object identifier processing
ospasn1object.c	ASN.1 object
ospasn1parse.c	ASN.1 parsing functions and look-up tables
ospasn1primitives.c	ASN.1 primitive types
ospaudit.c	Auditing functionality
ospauthcnf.c	AuthorisationConfirmation object
ospauthind.c	AuthorisationIndication object
ospauthreq.c	AuthorisationRequest object
ospauthrsp.c	AuthorisationResponse object
ospb64.c	Base64 processing
ospbfr.c	Buffer objects
ospbsafetstd.c	BSAFE wrapper functions
ospcallid.c	Call identifier processing
ospcisco.c	Support functions for Cisco extensions
ospcomm.c	Communications manager
ospcrypto.c	Generic cryptographic processing
ospcryptowrap.c	Cryptographic wrapper functions
ospdest.c	Destination object

OSP Toolkit Source Files	
osp enroll.c	Client enrollment functions
osp enrollutil.c	Client enrollment utility functions
osp fail.c	Failure reason processing
osp http.c	HTTP processing
osp init.c	Initialization
osp list.c	Linked list
osp mime.c	MIME processing
osp msgattr.c	Attribute processing
osp msgdesc.c	Message descriptor object
osp msgelem.c	Message element object
osp msginfo.c	Message information object
osp msgque.c	Message queue
osp msgutil.c	Message utility functions
osp nossl.c	SSL stub (for non-secure operation)
osp openssl.c	OpenSSL interface functions
osp ostime.c	Time-of-day processing
osp pkcs1.c	Public key cryptography standard 1 processing
osp pkcs7.c	Public key cryptography standard 7 processing
osp pkcs8.c	Public key cryptography standard 8 processing
osp provider.c	Provider object
osp providerapi.c	Provider interface functions
osp reauthreq.c	ReauthorisationRequest message element
osp reauthrsp.c	ReauthorisationResponse message element
osp secssl.c	Security SSL functions
osp security.c	Security functions
osp socket.c	Socket interface functions
osp ssl.c	SSL processing
osp statistics.c	Statistics processing
osp status.c	Status message element
osp tnaudit.c	Audit processing
osp tnlog.c	Logging
osp tnprobe.c	QoS probe functions
osp token.c	Token processing
osp tokeninfo.c	Token information object
osp trans.c	Transaction object
osp transapi.c	Transaction interface processing
osp usage.c	Usage reporting object
osp usagecnf.c	UsageConfirmation object
osp usageind.c	UsageIndication object
osp utils.c	Utility functions
osp version.c	Build version string
osp x509.c	X.509 certificate processing
osp xml.c	XML object
osp xmlattr.c	XML attribute object
osp xmlelem.c	XML element object
osp xmlenc.c	XML encoding functions
osp xmlparse.c	XML parsing functions
osp xmltype.c	XML character type processing

OSP Toolkit Source Files	
ospxmlutil.c	XML utility functions

Miscellaneous files

OSP Toolkit Miscellaneous Files	
makefile	UNIX make file
osp.def	Microsoft DDL export definitions file
osp.dsp	Microsoft Visual Studio 6.0 Project File
osp.dsw	Microsoft Visual Studio 6.0 Workspace File
ospcflags.inc	Common includes used by makefile
ospcompile.ksh	Unix OSP compilation script
Enroll.sh	Enrollment script

Building the OSP Toolkit

Supported compilers

As delivered, the TransNexus OSP Toolkit can be compiled for Windows NT 4.0, Solaris 2.7, or Solaris 2.8 using the following compilers.

- gcc/g++ 2.7.2.3
- SparcWorks C++ 5.0
- Microsoft Visual Studio Micro/Visual C++ 5.0 and 6.0 or Microsoft .NET

Compiling under Windows NT/Windows2000/Solaris

Please refer to the 'How to Build the OSP Toolkit' document for details on this.

Frequently Asked Questions

This section includes common questions about the OSP Toolkit, along with answers.

What Versions of OSP does the Toolkit Support?

Version 1.4.2 and 1.4.3, the official published version of the standard. (But see the next question for an exception.)

Why does the Toolkit spell Authorisation with an “s”?

Because of its origin within the European Telecommunications Standards Institute, the original OSP drafts used accepted International English spellings. In the final publication, however, ETSI revised the spellings to agree with American English. Because many OSP servers had been built and deployed based on earlier drafts, the client continues to use

the original (International) spellings. Newer servers are likely to support either spelling, so the Toolkit client achieves maximum interoperability by using the original spelling. As older servers are phased out, future revisions of the Toolkit will use the American spelling.

Is the Toolkit thread-safe?

Absolutely. In fact the Toolkit makes extensive use of threads to significantly improve performance.

Does the Toolkit support non-blocking calls?

Yes, it can; please contact TransNexus for details. Applications may also simulate non-blocking behavior by creating a separate thread for each call.

What is the default memory management scheme implemented in the toolkit and how does it affect the toolkit performance?

The toolkit, in previous release used malloc, the standard memory allocator available on Solaris, Windows and Unix platforms, as the default memory management tool. However, using a single-threaded malloc in a multithreaded application can degrade performance. As memory is being allocated concurrently in multiple threads, all the threads must wait in a queue while malloc() handles one request at a time. With a few extra threads, this can slow down performance, causing a problem known as heap contention. The toolkit may start experiencing this problem as the number of open transaction increases. To get over this problem, the 3.1.1 release of the toolkit/test_app has been modified to use mtmalloc for memory allocation. However, the toolkit has also been tested for performance optimizations with some other memory allocation schemes. A brief description of these schemes and the results is given below.

1. mtmalloc:

mtmalloc is a general-purpose allocator that is designed for use with multithreaded applications on Solaris. To use mtmalloc, the following changes need to be made in the source code:

- Add '-lmtmalloc' to the LFLAGS parameter in the test/Makefile file.
- Recompile the test_app.

The results compiled after testing the toolkit with mtmalloc showed a significant performance improvement in sending bulk authorization requests. The high number of open transactions did not seem to slow the memory allocation calls. However, it was observed that deleting the high number of transaction took more time with mtmalloc than malloc. Since transaction deletion is not a very time critical event, it is probably something that can be lived with. These tests were conducted on Solaris2.7.

2. Hoard:

Hoard is a fast, memory-efficient allocator that utilizes multiple processors. Hoard can dramatically improve the performance of multithreaded programs running on multiprocessors. It is a drop-in replacement for malloc and does not need a change in the source code. The following steps need to be executed to download and link toolkit with hoard:

On Solaris:

- Download libhoard.so from <http://www.hoard.org>.

- Copy the libhoard.so to /path/to directory.
- Set the environment variable LD_PRELOAD by executing this command:

```
Setenv LD_PRELOAD "path/to/libhoard.so
/usr/lib/libthread.so /usr/lib/librt.so
/usr/lib/libCrun.so.1"
```

On Linux:

- Download libhoard.so from <http://www.hoard.org>.
- Copy the libhoard.so to /path/to directory.
- Set the environment variable LD_PRELOAD by executing this command:

```
setenv LD_PRELOAD "/usr/lib/libdl.so
/path/to/libhoard.so /usr/lib/libpthread.so"
```

On Windows:

- Download libhoard.dll and libhoard.lib from <http://www.hoard.org>.
- Add the downloaded libraries in your library path.
- Type the following lines at the top of /include/osposincl.h file:

```
#if defined(USE_HOARD) && defined(_WIN32)
#pragma comment(lib, "libhoard.lib")
#endif
```

Pragma ensures that Hoard loads before any other library (you will need libhoard.lib in your path). When you execute your program, as long as libhoard.dll is in your path, your program will run with Hoard instead of the system allocator.

- Recompile the toolkit.

The results obtained after testing the toolkit with Hoard were very similar to the results obtained with mtmalloc. The improvement seen with hoard was that bulk deletion of transaction did not take as much time as it did with mtmalloc.

To improve the toolkit performance, either of the memory allocation schemes discussed above can be used. The user is recommended to make an evaluation as per the requirements.