TransNexus

OSP Toolkit

Porting Guide

Release 2.5.5

09 February 2002

OSP Toolkit

Porting Guide

Release 2.5.5

09 February 2002

Document 0300-1313-0200

TransNexus
1140 Hammond Drive, Building E
Suite 5250
Atlanta, GA 30328
USA

Phone:	+1 770 671 1888
Fax:	+1 770 671 1188

E-mail:	support@transnexus.com

Contents

Introduction

This document provides a porting guide for of release 2.5.5 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 eleven separate documents, including this one. The documents are:

- *Introduction*
- *Implementation Guide*
- *How to Build and Test the OSP Toolkit*
- *Errorcode List*
- *Programming Interface*
- *Cisco Interoperability Example*
- *Device Enrollment*
- *Internal Architecture*
- *Porting Guide*
- *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, 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,
- Thread support (either POSIX or Windows), 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, expect 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)
ospp	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 Toolkit Header Files	
`osp.h`	Public application programming interface and constants
`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
`osposssys.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

OSP Toolkit Header Files	
`ospsocket.h`	Socket interface
`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
`ospxmlelem.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

OSP Toolkit Source Files	
`ospdest.c`	Destination object
`ospenroll.c`	Client enrollment functions
`ospenrollutil.c`	Client enrollment utility functions
`ospfail.c`	Failure reason processing
`osphttp.c`	HTTP processing
`ospinit.c`	Initialization
`osplist.c`	Linked list
`ospmime.c`	MIME processing
`ospmsgattr.c`	Attribute processing
`ospmsgdesc.c`	Message descriptor object
`ospmsgelem.c`	Message element object
`ospmsginfo.c`	Message information object
`ospmsgque.c`	Message queue
`ospmsgutil.c`	Message utility functions
`ospnossl.c`	SSL stub (for non-secure operation)
`ospopenssl.c`	OpenSSL interface functions
`osppostime.c`	Time-of-day processing
`osppkcs1.c`	Public key cryptography standard 1 processing
`osppkcs7.c`	Public key cryptography standard 7 processing
`osppkcs8.c`	Public key cryptography standard 8 processing
`ospprovider.c`	Provider object
`ospproviderapi.c`	Provider interface functions
`ospreauthreq.c`	ReauthorisationRequest message element
`ospreauthrsp.c`	ReauthorisationResponse message element
`ospsecssl.c`	Security SSL functions
`ospsecurity.c`	Security functions
`ospsocket.c`	Socket interface functions
`ospssl.c`	SSL processing
`ospstatistics.c`	Statistics processing
`ospstatus.c`	Status message element
`osptnaudit.c`	Audit processing
`osptnlog.c`	Logging
`osptnprobe.c`	QoS probe functions
`osptoken.c`	Token processing
`osptokeninfo.c`	Token information object
`osptrans.c`	Transaction object
`osptransapi.c`	Transaction interface processing
`ospusage.c`	Usage reporting object
`ospusagecnf.c`	UsageConfirmation object
`ospusageind.c`	UsageIndication object
`osputils.c`	Utility functions
`ospversion.c`	Build version string
`ospx509.c`	X.509 certificate processing
`ospxml.c`	XML object
`ospxmlattr.c`	XML attribute object
`ospxmlelem.c`	XML element object
`ospxmlenc.c`	XML encoding functions
`ospxmlparse.c`	XML parsing functions

OSP Toolkit Source Files	
`ospxmltype.c`	XML character type processing
`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

Building the OSP Toolkit

Before attempting a port to a new target, it may be useful to build the Toolkit in one of its natively supported environments. This subsection lists the compilers that the Toolkit supports as delivered, and it describes how to compile the Toolkit source in Windows NT and Solaris environments.

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 /Visual C++ 5.0 and 6.0

Compiling under Windows NT

The OSP Toolkit includes a Visual C++ project file, `osp.dsw`, that automatically prepares the Visual Studio environment. Simply open that file from within Visual Studio to begin the build process.

Compiling under Solaris

The `ospcompile.ksh` script ensures the proper pre-compiler flags are set in order to build the desired OSP target library. It presents a simple menu with a choice of output options.

```
                  OSP Build Menu
            ----------------------------
            1. Client Production Library
```

```
         2. Client Development Library
         3. Client Non-Secure Library

         q. Quit
                   - - - - - - - - - - - - - - - -
                   Enter Selection =>
```

The output options determine compiler optimization settings, whether the Toolkit requires cryptographically strong authorization tokens, and whether or not special debugging code is included. The following table shows the options each menu choice selects.

	Production	Development	Non-Secure
Compiler Optimization	c	C	C
Cryptographic Tokens	c	c	C
Debugging Code	C	c	c
Legend: c enabled C disabled			

The `OSPVBuildVersion` global variable contains the attribute flags used to build the Toolkit target library as well as architecture environment and timestamp information. This variable is generated by `ospcompile.ksh` and inserted into the file `ospversion.c`.

Attribute Name	Description
`stok`	Secure Tokens enabled (PKCS#7 tokens expected)
`dbg`	OSP debug enabled. see `ospdebug.h` for more information on levels
`cdbg`	compiler debug enabled for code debugging.
`Hwe`	hardware acceleration support enabled. Optional - requires hardware acceleration devices and drivers.
`opt`	Optimized compile (Level 3 – O3).
`smime`	S/MIME enabled for message signing
`cdat`	use CDATA encoding only for binary XML element values
`gcc`	gcc used to compile `libosp.a`
`gpp`	g++ used to compile `libosp.a`
`spro`	SUN SparcWorks used to compile `libosp.a`
`gprf`	gprof profiler enabled
`clnt`	OSP client library target

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.