



OSP Toolkit

How to Build and Test the OSP Toolkit

Release 3.3.1

17 June 2005

Revision History

Revision	Date of Issue	Changes
2.7.0	March 12 th , 2003	Added revision history. Added documentation to support Linux during compilation. Added a section explaining the compilation procedure on Windows .NET.
2.8.0	March 20 th , 2003	Modified the section on enrollment to reflect the new enrollment procedure.
2.8.1	March 25 th , 2003	No Changes.
2.8.2	April 9 th , 2003	Added some FAQ's.
2.9.0	June 1 st , 2003	Updated the test_app CLI Interface to include the new API's implemented.
2.9.1	July 7 th , 2003	Added another FAQ. Removed documentation for pthread specific configuration on Windows.
2.9.2	July 28 th , 2003	Modified the Introduction to include the Interoperability documents.
2.9.3	Sep 15 th , 2003	Updated the test_app menu with the <i>RequestSuggestedAuthorization</i> API.
2.11.1	Feb 12 th , 2004	Added a new section on Enrollment script. Updated the FAQ as per the changes in the code for openssl initialization and token signing procedures. Removed comments on non-secure mode of compilation.
3.0	March 11 th , 2004	Updated the section on enrollment to reflect the change in the cacert.pem file name format.
3.1	April 8 th , 2004	Modified the testing procedure for test100.
3.1.1	May 12 th , 2004	No Changes
3.1.2	July 1 st , 2004	Updated the make procedure for the Toolkit
3.3.1	June 17 th , 2005	Modified the building procedure for Windows.

E-mail: support@transnexus.com

www.transnexus.com

Copyright © 2003-2005 by TransNexus. All Rights Reserved. OSP Secured is a trademark of TransNexus, Inc.

Contents

Revision History	2
Contents	3
Introduction	6
OSP Toolkit Primer	6
Building the Toolkit in Unix/ Linux	7
Introduction	7
Unpacking the Toolkit	8
Preparing to Build the OSP Toolkit	9
Building the OSP Toolkit	9
Building the Enrollment Utility	10
Building the Test_App Client Simulator	11
Building the Toolkit in Windows NT/2000	11
Introduction	11
Prepare Third Party Software and Unpack the OSP Toolkit Distribution	12
Obtaining a Pthreads Library and Supporting Files for Windows	12
Using the Visual C++ Compiler	13
Configure the OSP Toolkit Distribution for Windows	13
Compile the OSP Toolkit for Windows	13
Compile the Enrollment Utility for Windows	14
Compile the Test_App Utility for Windows	14
Using the Visual .NET Compiler	15
Configure the OSP Toolkit Distribution for Windows	15
Compile the Enrollment Utility for Windows	17
Compile the Test_App Utility for Windows	18
Testing the Toolkit	19
Overview	19
Preparations for Testing	19
Using the enroll script	20
Using the enroll program on Unix and Windows	21
Running the Client Simulator (test_app):	24

How to Integrate the OSP Toolkit with Your Application	28
Appendix A: Custom Compile Options	29
Introduction.....	29
Description of Compile Options	29
Custom Compilation in Unix.....	29
Custom Compilation in Windows.....	30
Frequently Asked Questions	31
In Windows , why do I receive a fatal error stating that a file cannot be found?.....	31
What are the differences between the TransNexus OSP Nexus Server and the TransNexus OSP Test Server?	31
I need more debugging information from the OSP Toolkit. How do I get it?	31
Why do I get the following errors when I send an Authorization Request?	31
If the OSP Test Server will not accept SSL messages, why do we have to use OpenSSL?.....	32
Why does the OSP client clock need to be synchronized with the OSP Server clock?32	
When compiling Enroll and Test_app in Windows, why do I get a large number of compiler errors similar to the following?.....	32
I received a certificate, but there are errors. Did I successfully enroll?	35
While compiling the OSP Toolkit in Windows, I received approximately 94 errors and 65 warnings all in ospcryptowrap.c and in ospopenssl.c. Why does this happen?	35
Why do I get the following error while trying to obtain a certificate request and private key (when enrolling)?.....	35
I am trying to connect to the OSP Server, but the request just blocks holding up all my resources. What is wrong?.....	36
Why will my application not establish SSL connection?	36
What hardware devices are supported?.....	36
I don't have a cryptographic hardware acceleration device, what do I do?	36
My version of OpenSSL does not have Engine API, and I get compile time errors. What do I do?.....	36
How do I know if my cryptographic hardware acceleration device is being used?	37
I have a cryptographic hardware acceleration device but it is NOT being used, why?37	
I have more than one supported cryptographic hardware acceleration device how do I know which one will be used?.....	37
What can a cryptographic hardware acceleration device be used for?	37
Why do I get errors saying 'undefined symbols' when I try to build the toolkit?	37
What values should I configure for - HttpRetryLimit, HttpRetryDelay and HttpRetryTimeOut ?.....	37

What encryption mechanisms are supported in the toolkit ?..... 38

E-mail: support@transnexus.com
www.transnexus.com
Copyright © 2003 by TransNexus. All Rights Reserved.

Introduction

This document provides build instructions for of release 3.3.1 of the Open Settlement Protocol (OSP) Toolkit. That Toolkit, which is freely available 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*
- *OSP Toolkit Errorcode List*
- *Programming Interface*
- *Cisco Interoperability*
- *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 reader is advised to read the *Introduction*, *H.323 Implementation Guide* and *SIP Implementation Guide* for an overview of OSP before using this document. The Toolkit has been tested with following operating systems and compilers: Solaris 2.7 and Solaris 2.8 (using GNU GCC 2.95.3 and SparcWorks C++ 5.0), Red Hat Linux 8.0 and Fedora Core 3 SELinux (using GNU GCC 3.4.2), Windows NT (using Visual Studio .NET C++ 7.0 compiler) and Windows 2000 (using Visual C++ 6.0 compiler).

This document is the result of our customers' questions regarding the OSP Toolkit. We thank our customers for their input, and we thank you for your interest in our software.

OSP Toolkit Primer

The OSP Toolkit, when compiled, is a library comprised of functions that simplifies sending and receiving OSP messages. It is this library, which will be integrated into your software. The OSP Toolkit uses third party software (by default OpenSSL) for cryptographic algorithms and for secure internet transactions (HTTPS). Two additional applications provide examples of how the OSP Toolkit is to be used. The first, enroll, is an example of how to add your device to an OSP Server using the TEP protocol. You can use this application, or create your own following the guidelines within the document *Device*

Enrollment. Test_app, the other application, allows you to test the OSP Toolkit once it has been compiled; it also provides an example of how the OSP Toolkit can be used. The relationship between all of these software components is illustrated below, in Figure 1.

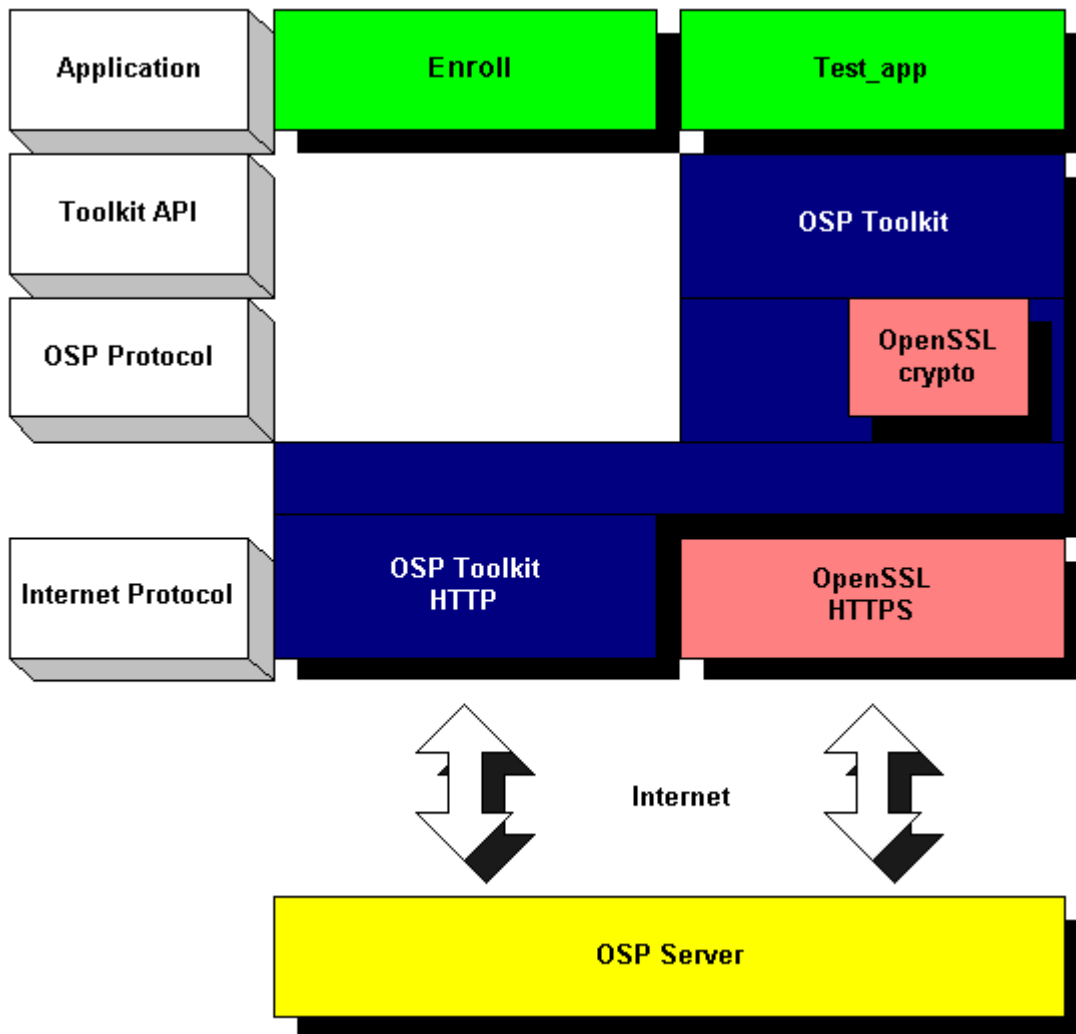


Figure 1: Structure and Layers of the OSP Toolkit

Building the Toolkit in Unix/ Linux

Introduction

The procedure for building the Toolkit is same in both Linux as well as Unix. However, it is slightly different for building the Enrollment utility and the test application. In order to successfully compile and use the OSP Toolkit, the following list contains all the software used by the OSP Toolkit:

OpenSSL (required) - Open Source SSL protocol and Cryptographic Algorithms (version 0.9.7g recommended) from www.openssl.org. Pre-compiled OpenSSL packages are not recommended because of the binary compatibility issue.

Perl (required) - A programming language used by OpenSSL for compilation. Any version of Perl will work. One version of Perl is available from www.activestate.com/ActivePerl. If pre-compiled OpenSSL packages are used, Perl package is not required.

C compiler (required) - Any C compiler should work. For example, we have successfully used the GNU Compiler Collection from www.gnu.org

OSP Server (recommended for testing) - Access to any OSP server should work. Instructions below describe how to test for free with the TransNexus OSP Test Server.

Unpacking the Toolkit

After downloading the OSP Toolkit from www.transnexus.com, perform the following steps in order:

- 1) Copy the OSP Toolkit distribution into the directory where it will reside.
- 2) Unzip the tar file by executing the following command:

```
gunzip OSPToolkit-###.tar.gz
```

Where **###** is the version number separated by underlines. For example, if the version is 2.5.4, then the above command would be:

```
gunzip OSPToolkit-2_5_4.tar.gz
```

A tar file will replace the *.gz file.

- 3) Untar the file by executing the following command:

```
tar -xvof OSPToolkit-###.tar
```

Where **###** is the version number separated by underlines. For example, if the version is 2.5.4, then the above command would be:

```
tar -xvof OSPToolkit-2_5_4.tar
```

A new directory (`osptoolkit`) will be created within the same directory as the tar file.

- 4) Go to the `osptoolkit` directory by running this command:

```
cd osptoolkit
```

Within this directory, you will find directories and files similar to what is listed below (if the command "ls -F" is executed):

```
> ls -F
enroll/
RelNotes.txt
README.txt
bin/
crypto/
include/
lib/
license.txt
src/
test/
```

Preparing to Build the OSP Toolkit

- 5) Compile OpenSSL according to the instructions provided with the OpenSSL distribution. Also, make sure the header files and library files are not in a shared directory, such as `/usr/local/`.
- 6) Copy the OpenSSL header files (ones with the `*.h` extension contained in the `include/openssl` directory within the directory containing OpenSSL files into the `crypto/openssl` directory within the Toolkit. The OpenSSL header files are located under the `include/openssl` directory.
- 7) Copy the OpenSSL library files (`libcrypto.a` and `libssl.a`) into the `lib` directory within the Toolkit. The OpenSSL library files are located within the `openssl` directory.

Building the OSP Toolkit

- 8) From within the OSP Toolkit directory, start the compilation script by executing the following commands:

```
cd src;

make clean; make build
```

The make script can also be used to install the toolkit header files and the toolkit library in the folder that the user wishes to install them in. The default path is: `/usr/local/`. The user can modify this to any path desired. Also, the "make install" command should be used to install the files.

By default the toolkit is compiled in the production mode. The following table identifies which default features are activated with each compile option:

Default Feature	Production	Development
Debug Information Displayed	No	Yes
* Contact TransNexus for more information about cryptographic hardware acceleration. See Appendix A for more information and/or custom compile options.		

The "Development" option is recommended for a first time build. The "CFLAGS" definition in the Makefile would need to be modified to build in development mode. If you need to customize your build, see Appendix A for details of how to accomplish this. Compilation is successful if there are no errors anywhere in the compiler output.

Note: Token Signing is now made a user option. For each token that is being validated, the application will now tell the toolkit if the token is signed or unsigned. The toolkit then validates the token using the appropriate algorithm. If the application is not sure of what algorithm to use, it can ask the toolkit to try both the algorithms. Similarly, hardware acceleration is now a configurable parameter in the OSPPIInit API. See the Programming Interface document for details on these two changes.

Building the Enrollment Utility

Device enrollment is the process of establishing a trusted cryptographic relationship between the VoIP device and the OSP Server. The Enroll program is a sample application for a TEP client. For more details about how to create a TEP client, see the document "Device Enrollment" (ToolKit Enrollment.pdf). The enroll program may be used to add your device to a TransNexus OSP Test Server; otherwise, enrollment can be bypassed by using the certificates contained in the confirmation message if you register to use the OSP Test Server. See the section below "Using the Enroll Program to Enroll a Device".

- 9) From within the OSP Toolkit directory, execute the following commands at the Unix command prompt:

```
cd enroll  
  
make clean; make
```

Compilation is successful if there are no errors anywhere in the compiler output. The enroll program is now located in the OSP Toolkit "bin" directory.

Compiling in Linux: From within the OSP Toolkit directory, execute the following commands at the Linux command prompt:

```
cd enroll  
  
make clean; make linux
```

Compilation is successful if there are no errors anywhere in the compiler output. The enroll program is now located in the OSP Toolkit "bin" directory.

Building the Test_App Client Simulator

Test_app is a client simulator to ensure that the Toolkit has been compiled properly.

10) From within the OSP Toolkit directory, execute the following commands at the Unix command prompt:

```
cd test  
  
make clean; make
```

Compilation is successful if there are no errors anywhere in the compiler output. The test_app program is now located within the OSP Toolkit "bin" directory.

Compiling in Linux: From within the OSP Toolkit directory, execute the following commands at the Linux command prompt:

```
cd test  
  
make clean; make linux
```

Compilation is successful if there are no errors anywhere in the compiler output. The test_app program is now located in the OSP Toolkit "bin" directory.

By this point, a fully functioning OSP Toolkit should have been successfully compiled in Unix/Linux.

Building the Toolkit in Windows NT/2000

Introduction

In order to successfully compile and use the OSP Toolkit, the following list contains all the software used by the Windows version of the OSP Toolkit:

OpenSSL (required) – Open Source SSL protocol and Cryptographic Algorithms (version 0.9.7g recommended) from www.openssl.org

Pthreads Library (required) - A standard Unix POSIX Multithreading library ported to Windows. This could be obtained from anywhere, but we recommend the one from [ftp://sources.redhat.com/pub/pthreads-win32/dll-latest](http://sources.redhat.com/pub/pthreads-win32/dll-latest)

C compiler (required) - Any C compiler should work. All the examples in this document are based on Visual C++ 6.0 and Visual .NET. For details purchasing either of these compilers, please visit www.microsoft.com.

OSP Server (recommended for testing) - Access to any OSP server should work. Instructions below describe how to test for free with the TransNexus OSP Test Server.

Prepare Third Party Software and Unpack the OSP Toolkit Distribution

- 1) Make sure that a C compiler is already installed on the machine. If it is not installed, install it.
- 2) Download and install Win32 OpenSSL v0.9.7g binary distribution from www.openssl.org.
- 3) Copy the OSP Toolkit gzipped file to the desired directory.
- 4) Unzip the tar file by using any application that unzips and untars files. The unzipped files may be placed in any desired directory.
- 5) Copy the OpenSSL header files (ones with the *.h extension) into the crypto\openssl directory within the Toolkit. The OpenSSL header files are located under the <openssl_path>\include\openssl directory.
- 6) Copy the OpenSSL library files ssleay32.lib and libeay32.lib from the <openssl_path>\lib\VC directory into the lib directory within the Toolkit.

Obtaining a Pthreads Library and Supporting Files for Windows

- 7) Create "pthread" directory on the same level with the "src" directory.

```
mkdir pthread
```

- 8) Create two other directories within the new directory:

```
cd pthread
mkdir include
mkdir lib
cd ..\..
```

- 9) Download a pre-compiled Win32 pthreads library from the following website (or similar web site):

<ftp://sources.redhat.com/pub/pthreads-win32/prebuilt-dll-2-7-0-release>

These files will be placed into the "pthread" directory.

10) Within the "lib" directory (on the above web site) download the following files:

pthreadVC2.lib (place this in the "pthread\lib" directory within the Toolkit)

pthreadVC2.dll (place this in the "bin" directory within the Toolkit)

11) Download the entire contents of the include directory and place them in the pthread\include directory within the Toolkit.

Now, these different projects (OSP Toolkit, Enroll, and Test_App) need to be cleaned and recompiled.

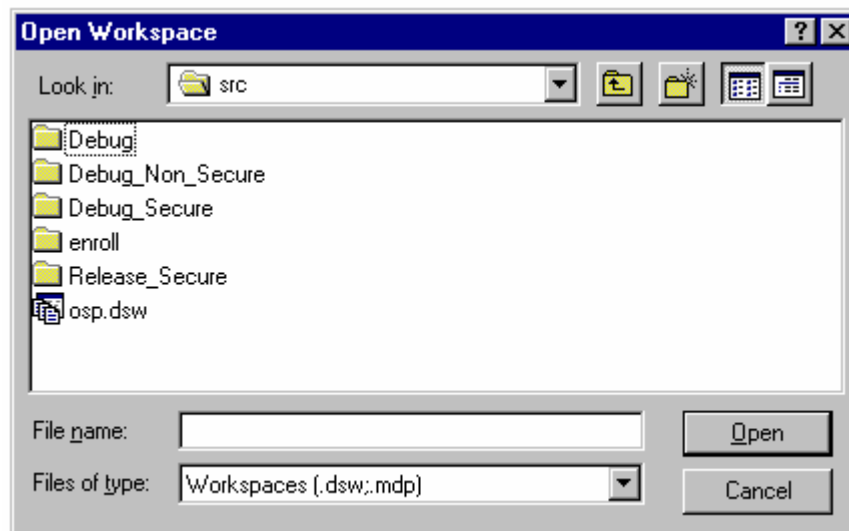
Using the Visual C++ Compiler

Configure the OSP Toolkit Distribution for Windows

12) Open the Visual C++ project: Run the Visual C++ Program.

Menu: File -> OpenWorkspace

13) Within the directory containing the Toolkit, go to the src directory and select osp.dsw and hit the "Open" button.



Compile the OSP Toolkit for Windows

Build the OSP Toolkit within the Visual C++ environment.

14) Set the active project to "osp":

Menu: Project -> Set Active Project -> osp

Menu: Build -> Set Active Configuration...

Set the active configuration to one of the three listed in the table below:

Default Feature	Release	Debug
Debug Information Displayed	No	Yes
* Contact TransNexus for more information about cryptographic hardware acceleration. See Appendix A for more information and/or custom compile options.		

15) Compile the Toolkit. Menu: Build -> Build osp.lib

Compile the Enrollment Utility for Windows

16) Set the active project to "enroll" Menu:

Project -> Set Active Project -> enroll

17) Select the active configuration to one that matches the configuration of the compiled OSP Toolkit (see the table below).

Build -> Set Active Configuration...

If Toolkit Compile Option Was...	Enroll Active Configuration Should Be...
Release	Release
Debug_Secure	Debug

18) Compile the Enroll utility.

Menu: Build -> Build Enroll.exe

Compile the Test_App Utility for Windows

Build the test_app utility, and run the program:

19) Set the active project to "test_app"

Menu: Project -> Set Active Project -> test_app.exe

20) Set the active configuration to one that matches the configuration of the compiled OSP Toolkit (see the table below).

21) Menu: Build -> Set Active Configuration...

If Toolkit Compile Option Was...	Test_App Active Configuration Should Be...
Release	Win32 Release
Debug_Secure	Win32 Debug

22) Compile test_app.

Menu: Build -> Build Test_app.exe

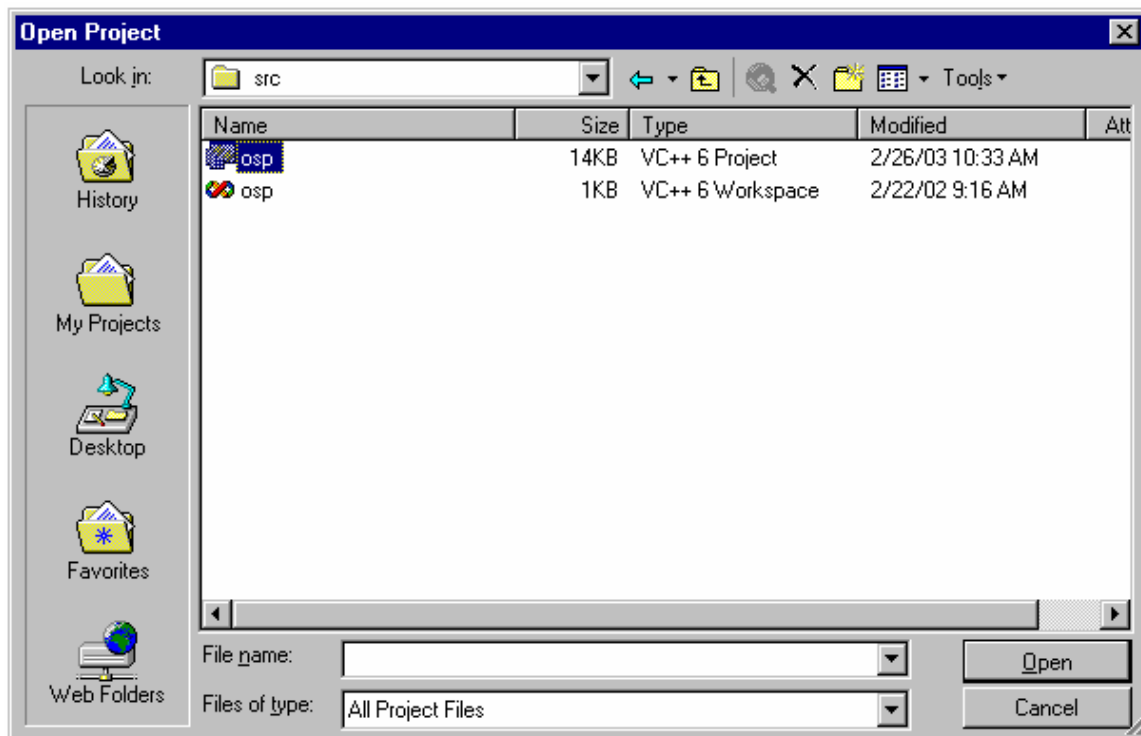
Using the Visual .NET Compiler

Configure the OSP Toolkit Distribution for Windows

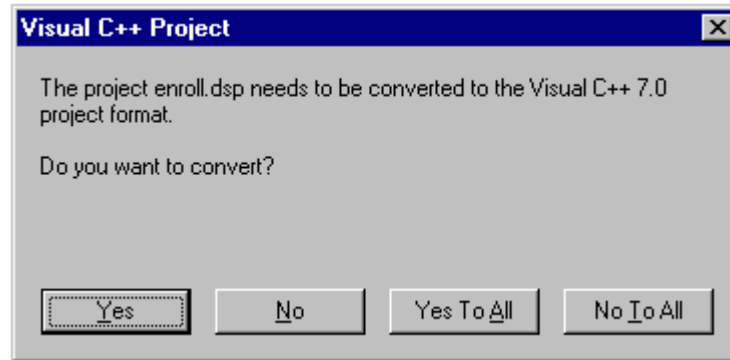
23) Open the Visual .NET project: Run the Visual .NET Program.

Menu: File -> Open -> Project...

24) Within the directory containing the Toolkit, go to the src directory and select osp.dsw (the VC++ 6 Workspace file) and click the "Open" button.



You might be asked to convert the projects, like in the following dialog. Click "Yes To All".



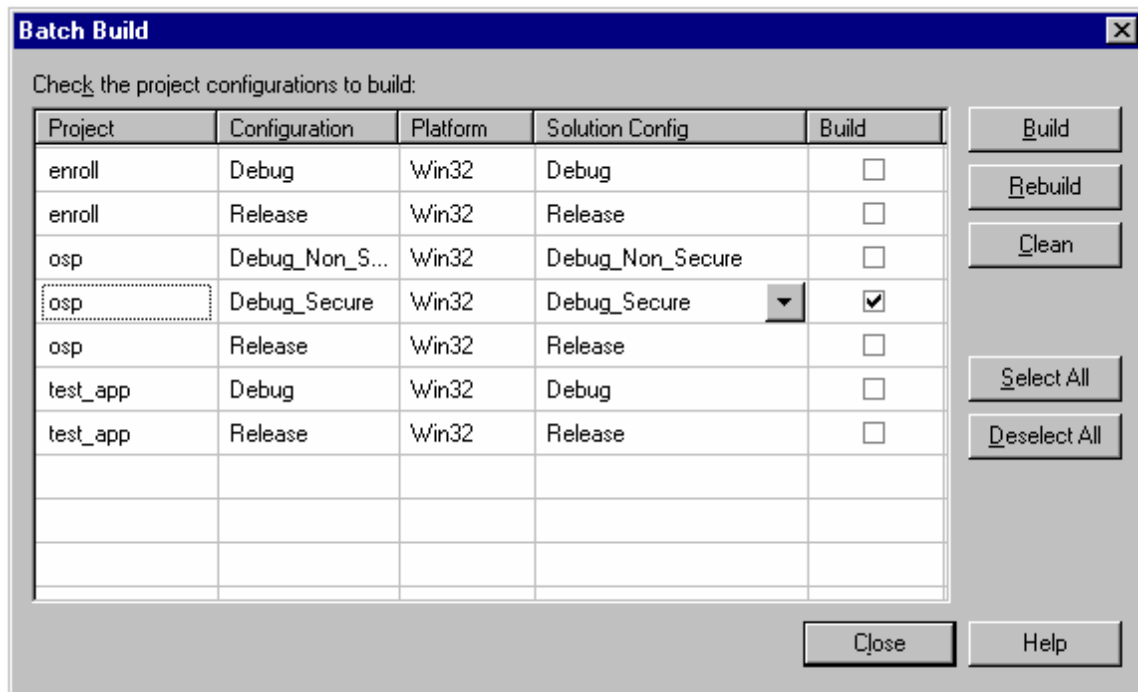
Compile the OSP Toolkit for Windows

Build the OSP Toolkit within the Visual C++ environment.

25) Go to the Batch Build dialog box:

Menu: Build -> Batch Build...

A similar dialog box to the example below should appear.



Set the build checkmark for the osp Toolkit configuration to one of the three listed in the table below (the one in the example has the build set to "Debug_Secure"):

Default Feature	Release	Debug
Debug Information Displayed	No	Yes
* Contact TransNexus for more information about cryptographic hardware acceleration. See Appendix A for more information and/or custom compile options.		

26) Clean the OSP Project

Click the "Clean" Button.

27) Compile the OSP Toolkit

Menu: Build -> Batch Build...

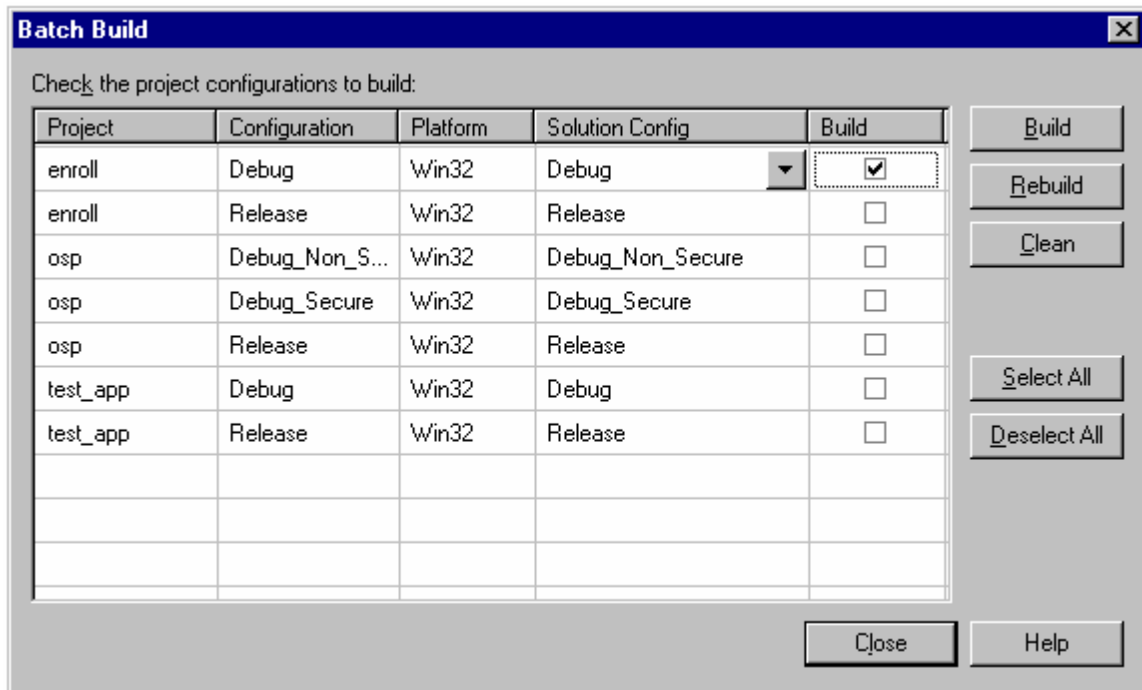
Click the "Build" Button.

Compile the Enrollment Utility for Windows

28) Go to the Batch Build dialog box:

Menu: Build -> Batch Build...

A similar dialog box to the example below should appear.



Set the build checkmark for the enroll configuration to one of the three listed in the table below (the one in the example has the build set to "Debug"):

If Toolkit Compile Option Was...	Enroll Active Configuration Should Be...
Release	Release
Debug_Secure	Debug

29) Clean the Enroll Project

Click the "Clean" Button.

30) Compile the Enroll program:

Menu: Build -> Batch Build...

Click the "Build" Button.

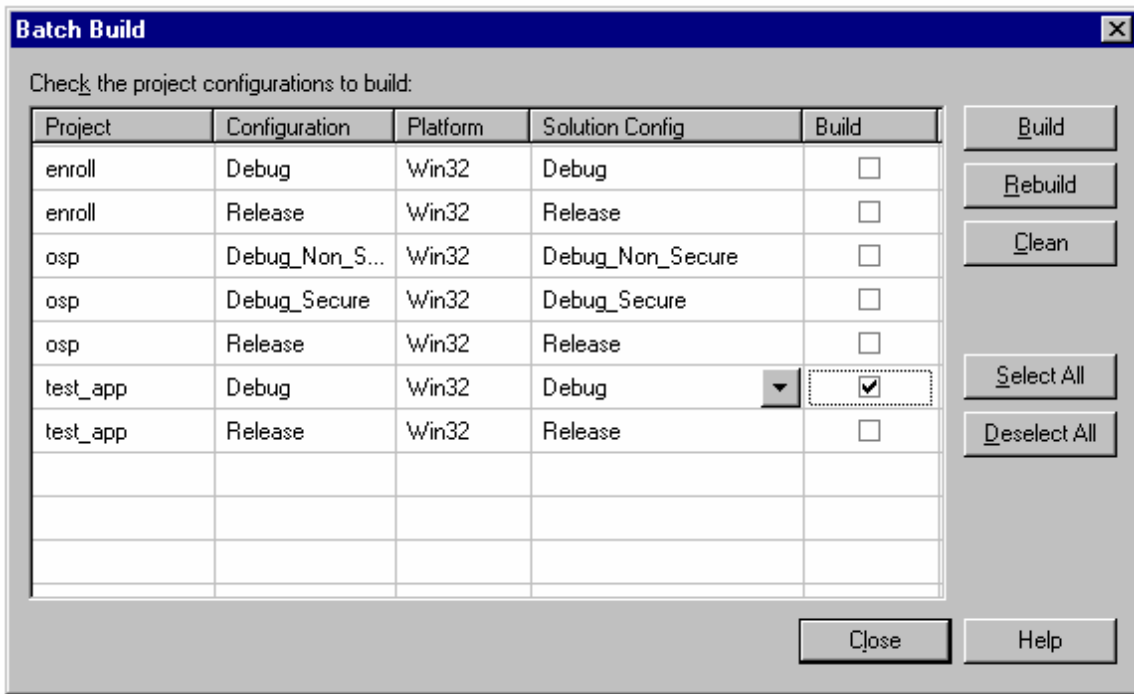
Compile the Test_App Utility for Windows

Build the test_app utility, and run the program:

31) Go to the Batch Build dialog box:

Menu: Build -> Batch Build...

A similar dialog box to the example below should appear.



Set the build checkmark for the test_app configuration to one of the three listed in the table below (the one in the example has the build set to "Debug"):

If Toolkit Compile Option Was...	Test_App Active Configuration Should Be...
Release	Release
Debug_Secure	Debug

32) Clean the Test_app Project

Click the "Clean" Button.

33) Compile the Test_app program:

Menu: Build -> Batch Build...

Click the "Build" Button.

Testing the Toolkit

Overview

Test_app is used to simulate a client application such as a VoIP gateway, gatekeeper, SIP proxy or softswitch. For example, test_app makes calls to the OSP Toolkit API that request an IP address of a destination network that can complete a call to the called number. The OSP Toolkit sends the request to an OSP Server and then returns the response from the OSP server to test_app. This section provides instructions of how to use test_app to call specific OSP Toolkit API functions.

Preparations for Testing

Before testing and verifying the compiled OSP Toolkit, the client simulator (test_app) must be configured. Test_app is pre-configured for use with the TransNexus OSP Test Server. If the configuration of test_app needs to be modified (for example, use a different OSP server, change the number of destinations returned, etc.), then the test_app configuration file, "test.cfg," needs to be reconfigured. It is contained within the "bin" directory in the OSP Toolkit directory. Instructions are included within the test.cfg file to guide you through the configuration changes. To comment out lines in this file, precede each line with a "#". The test_app needs to be configured for the Source IP address/ Source device IP Address.

After test_app has been configured, the next step is to enroll test_app with the OSP Server which will be used for testing. Device enrollment is the process of establishing a trusted cryptographic relationship between the VoIP device (test_app) and the OSP Server. Refer to your OSP server documents for more instructions on how to enroll a device. Or, follow the instructions in the following section "Using the Enroll Program to Enroll Your Device" to enroll test_app with the TransNexus OSP Test Server which may be accessed for free via the Internet. Test_app expects three files with specific file names to exist in the bin directory. This required information is outlined in the table below.

File Type	File Name for use with test_app
Private Key	Pkey.pem
Local Certificate	Localcert.pem
CA Certificate	Cacert_0.pem

In order to access the TransNexus OSP Test Server, you must register your device with us. This is accomplished by sending an e-mail to support@transnexus.com with the following information:

- The name of at least one person we can contact from your company and his/her e-mail address.
- If the machine running test_app and/or using the OSP Toolkit is outside all firewalls, then we need the IP address of this machine.
- If the machine running test_app and/or using the OSP Toolkit is behind at least one firewall, then we need the IP address of the OUTERMOST firewall which the OSP messages will travel through (our test server must be able to see the machine with that address).

Within one business day of receiving your e-mail, we will send you an e-mail reply indicating that your device is configured with the TransNexus OSP Test Server. Included in this e-mail will be certificates to use with the TransNexus OSP Test Server. These certificates (private key, local certificate, and self-signed CA certificate) should be placed in the "bin" directory within the OSP Toolkit directory. If you wish to use or generate your own certificates, see the next section for details. For first time users of the OSP Toolkit, we recommend starting with the default configuration and use the certificates we give you, then working toward more complexity.

NOTE: The OSP Test Server is configured only for sending and receiving non-SSL messages, and issuing signed tokens.

Using the enroll script

The OSP Toolkit includes an enrollment script – enroll.sh that the user can use to enroll with an OSP Server. The script takes the domain name or IP addresses of the OSP Servers that the toolkit needs to enroll with, and then generates .pem files – cacert_#.pem, certreq.pem, localcert.pem, and pkey.pem. The '#' in the cacert file name is used to differentiate the ca certificate file names for the various SP's. If only one address is provided at the command line, cacert_0.pem will be generated. If 2 addresses are provided at the command line, 2 files will be generated – cacert_0.pem and cacert_1.pem, one for each SP. The example below shows the usage when the client is registering with two OSP Servers. If all goes well, the following text will be displayed. The gray boxes indicate required input.

```
> enroll.sh 172.16.4.18 ospstestserver.transnexus.com

Using configuration from /usr/local/bin/openssl.cnf
Loading 'screen' into random state - done
Generating a 512 bit RSA private key
.....+++++
.....+++++
writing new private key to 'pkey.pem'
```

```

-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]: ████████
State or Province Name (full name) [Some-State]: ████████
Locality Name (eg, city) []: ████████
Organization Name (eg, company) [Internet Widgits Pty Ltd]: ████████
Organizational Unit Name (eg, section) []: ████████
Common Name (eg, YOUR name) []: ████████
Email Address []: ████████

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []: ████████
An optional company name []: ████████
Error Code returned from openssl command : 0
CA certificate received
[SP: 172.16.4.18]Error Code returned from getcacert command : 0

output buffer after operation: operation=request
output buffer after nonce: operation=request&nonce=1424260071824430
X509 CertInfo context is null pointer
Unable to get Local Certificate
depth=0 /CN=betarel.transnexus.com/O=OSPServer
verify error:num=18:self signed certificate
verify return:1
depth=0 /CN=betarel.transnexus.com/O=OSPServer
verify return:1
The certificate request was successful.
Error Code returned from localcert command : 0

CA certificate received
[SP: ospstestserver.transnexus.com]Error Code returned from getcacert
command : 0

```

If the user does not want to use the enrollment script, the user can follow the procedures in the following section to generate the crypto files individually.

NOTE: The enrollment script works only for Solaris and Linux. It does not work for Windows. The script expects the openssl.cnf file and the .rnd file to be present in the same directory as the enroll script. You can either copy the configuration file to the location or change the path in the enrollment script.

Note for Linux Users: The script *'enroll.sh'* requires AT&T korn shell (ksh) or any of its compatible variants.

Using the enroll program on Unix and Windows

If you simply want to use the certificates we have given you, this section may be skipped; continue with the section "Running the Client Simulator (test_app)". If you want to use your own certificates, then follow the instructions in this section. **In either case, we would need your device IP Address. You are thus required to send the e-mail mentioned above irrespective of which enrollment mechanism you use.**

Go to the bin directory within the OSP Toolkit directory.

```
cd bin
```

1) Generate a private key and a certificate request.

Execute the following command at the prompt:

```
openssl req -outform PEM -nodes -newkey rsa:512 -md5 -new
-out certreq.pem -keyform PEM -keyout pkey.pem
```

NOTE: For Windows, you must append the following text to the above command:

```
-config <openssl_path>\bin\openssl.cnf
```

where <openssl_path> is the location of your OpenSSL directory.

NOTE: On Windows, even if the configuration file name might be "openssl" instead of openssl.cnf, you MUST use "openssl.cnf" with the "-config" option.

If all goes well, the following text will be displayed. The gray boxes indicate required input.

```
Using configuration from /usr/local/bin/openssl.cnf
Loading 'screen' into random state - done
Generating a 512 bit RSA private key
.....+++++
.....+++++
writing new private key to 'pkey.b64'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]: ██████████
State or Province Name (full name) [Some-State]: ██████████
Locality Name (eg, city) []: ██████████
Organization Name (eg, company) [Internet Widgits Pty Ltd]: ██████████
Organizational Unit Name (eg, section) []: ██████████
Common Name (eg, YOUR name) []: ██████████
Email Address []: ██████████

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []: ██████████
An optional company name []: ██████████
```

The two files generated are base64-encoded files: the certificate request "certreq.pem" and the private key "pkey.pem". The public key is contained in the certificate request.

2) Get the Certificate Authority (CA) certificate from the server:

Execute the following command at the prompt:

```
enroll -function getcacert -caurl  
http://osptestserver.transnexus.com:1080/tep > cacert_#.pem
```

If you are using a different OSP server that supports TEP enrollment, you would put the server name in place of osptestserver.transnexus.com.

If there are no errors, a cacert_#.pem file would be generated. Check to make sure the certificate is in the proper format. The contents of the file should comprise only of the characters from "a" through "z", "A" through "Z", "0" through "9", "+", and "/". Do this step again if unsuccessful.

If the user is enrolling with only one SP, then # needs to be replaced with 0. The '#' in the cacert file is a digit (0, 1, 2, 3 ...) that uniquely identifies the files containing the different ca certificates. For multiple SP's, this command needs to be run for every SP. If the user wants to enroll with two different SP's, this command needs to be run twice, once with the#=0, and then with#=1. For the first SP, this number should be 0, and for ***every SP after that this number should increase by one.***

3) Enroll the device and receive a local certificate.

Execute the following command at the prompt:

```
enroll -function request -username trans -password nexus  
-customer 1000 -device 1000 -cacert cacert_0.pem -certreq  
certreq.pem -sslurl  
https://osptestserver.transnexus.com:1443/tep > localcert.pem
```

If you are using a different OSP server that supports TEP enrollment, put the server name in place of osptestserver.transnexus.com.

When you run this command, you may see the following, which may be disregarded.

```
output buffer after operation: operation=request  
output buffer after nonce:  
operation=request&nonce=1638481482129417  
X509 CertInfo context is null pointer  
Unable to get Local Certificate  
depth=0 /CN=osptestserver.transnexus.com/O=OSPServer  
verify error:num=18:self signed certificate  
verify return:1  
depth=0 /CN=osptestserver.transnexus.com/O=OSPServer  
verify return:1  
The certificate request was successful.
```

If there are no errors, the file localcert.pem contains the resulting certificate.

Open the file localcert.pem and check to make sure the certificate is in the proper format. The contents of the file should be comprised only of the characters from "a" through "z", "A" through "Z", "0" through "9", "+", and "/". Do this step again if unsuccessful.

After following the instructions for enrolling a TEP device, three files should exist: a private key, a local certificate, and a CA certificate. In order to use these files with test_app, they must be located in the bin directory within the OSP Toolkit directory.

File Type	File Name for use with test_app
Private Key	pkey.pem
Local Certificate	localcert.pem
CA Certificate	Cacert_#.pem

Running the Client Simulator (test_app):

Once your machine is configured and registered to a OSP server, follow these steps:

1) Run test_app by executing the following lines of code within the OSP Toolkit directory:

```
cd bin
test_app
```

The following text menu will be displayed:

```
Provider API functions
-----
1) New                               2) Delete
3) For future Enhancements          4) SetServicePoints
5) GetHTTPMaxConnections            6) SetHTTPMaxConnections
7) GetHTTPPersistence               8) SetHTTPPersistence
9) GetHTTPRetryDelay                10) SetHTTPRetryDelay
11) GetHTTPTimeout                  12) SetHTTPTimeout
13) For future Enhancements          14) SetCapabilitiesURLs
15) GetLocalValidation              16) SetLocalValidation
17) GetServicePoints                18) SetServicePoints
19) GetSSLLifetime                  20) SetSSLLifetime
21) GetNumberOfAuthorityCertificates 22) GetNumberOfServicePoints
-----
Transaction API functions
-----
23) New                               24) Delete
25) AccumulateOneWayDelay            26) AccumulateRoundTripDelay
27) GetFirstDestination              28) GetNextDestination
29) RequestAuthorisation             30) RequestSuggestedAuthorization
31) ValidateAuthorisation            32) ReportUsage
33) TransactionInitializeAtDevice(OGW) 34) TransactionInitialize(TGW)
35) SetNetworkId                    36) TransactionRecordFailure
37) IndicateCapabilities             38) RequestReauthorization
-----
Miscellaneous Tests
-----
39) GetDestinationProtocol           40) IsDestOSPEntered
41) 500 Test Calls                   42) Not implemented
43) BuildUsageFromScratch(OGW)       44) BuildUsageFromScratch(TGW)
45) GetLookAheadInfoIfPresent        46) ModifyDeviceIdentifiers
99) Sleep for 2 seconds
-----
Configuration Parameters
-----
```

```

50) Set Calling Number          51) Set Called Number
52) Get Calling Number         53) Get Called Number
-----
Performance tests
-----
100) Run Multiple calls
101) Run Multiple Capabilities Indications
-----
Enter function number or 'q' to quit =

```

* Same as 35 but sets a different value. Calling this is equivalent to resetting the network id with a different value.

Use the following sequence of function numbers to test OSP ToolKit API functions used to originate a call and terminate a call. Type each function number, hit the "enter" key to enter the function number, and then hit the "enter" key a second time at the "press any key to continue..." prompt to complete the API function.

```

1                               To establish new provider

23, 29, 27, 28, 32, 24        Simulate call origination:
                               Request destination(s), authorization
                               tokens and report usage

23, 29, 27, 31, 32, 24        Simulate call termination:
                               Validate token and report usage

2                               Delete provider

```

All functions should return zero for an error code. (Note that function number 28 will correctly return an error if another destination is not available.)

Enter "q" to exit the program.

The following is a list of expected results using test_app.

Note: Token validation will fail if the clock of test_app and <http://osptestserver.transnexus.com> are not synchronized. The time clock of the computer running test_app must be synchronized with a public NTP (network time protocol) server. The <http://osptestserver.transnexus.com> server clock is set to GMT (without any offsets for time zone).

Below, in *courier new font*, are the expected test_app results when using the default configuration file, test.cfg, and the TransNexus OSP Test Server, <http://osptestserver.transnexus.com>.

Provider API Function

1) New:

```

Enter function number or 'q' to quit => 1
Loaded 1 CA certificates
Loaded 1 local certificates
Loaded 1 private keys
function return code = 0
press any key to continue...

```

Transaction API Functions to simulate call origination

23) New:

```
Enter function number or 'q' to quit => 23
function return code = 0
press any key to continue...
```

29) RequestAuthorization

```
Enter function number or 'q' to quit => 29
num dest = 1
function return code = 0
press any key to continue...
```

27) GetFirstDestination

```
Enter function number or 'q' to quit => 27
callid size = 1 value = <31>
[1.1.1.1]
function return code = 0
press any key to continue...
```

28) GetNextDestination

```
Enter function number or 'q' to quit => 28
callid size = 1 value = <32>
[2.2.2.2]
function return code = 0
press any key to continue...
```

32) ReportUsage

```
Enter function number or 'q' to quit => 32
Reporting Usage for OSPVTransactionHandle 0
function return code = 0
press any key to continue...
```

24) Delete

```
Enter function number or 'q' to quit => 24
OSPVTransactionHandle deleted.
function return code = 0
press any key to continue...
```

Transaction API Functions to simulate call termination

23) New:

```
Enter function number or 'q' to quit => 23
function return code = 0
press any key to continue...
```

29) RequestAuthorization

```
Enter function number or 'q' to quit => 29
num dest = 2
function return code = 0
press any key to continue...
```

27) GetFirstDestination

```
Enter function number or 'q' to quit => 27
callid size = 1 value = <31>
```

```
[1.1.1.1]
function return code = 0
press any key to continue...
```

31) ValidateAuthorization

```
Enter function number or 'q' to quit => 31
CN:0;1%0#Uosptestserver.transnexus.com10U
  OSPServer
HN:osptestserver.transnexus.com
authorised = 1
function return code = 0
press any key to continue...
```

32) ReportUsage

```
Enter function number or 'q' to quit => 32
Reporting Usage for OSPVTransactionHandle 0
function return code = 0
press any key to continue...
```

24) Delete

```
Enter function number or 'q' to quit => 24
OSPVTransactionHandle deleted.
function return code = 0
press any key to continue...
```

Provider API Function

2) Delete:

```
Enter function number or 'q' to quit => 2
function return code = 0
press any key to continue...
```

2) Run `test_app` again to test multithreading by executing the following line of code:

```
test_app
```

Use the following sequence of numbers to test 1,000 calls.

```
100, 1, 1000, q
```

Type each function number, hit the "enter" key to enter the function number, and then hit the "enter" key a second time at the "press any key to continue..." prompt to complete the API function.

Enter "q" to exit the program.

NOTE: If test 100 is performed using <http://osptestserver.transnexus.com>, expect the test to take several minutes to complete. This OSP Test Server is a shared machine used by many developers for functional testing. The OSP Test Server is not configured for performance or volume testing.

NOTE: If you want to report the Network Identifier in the Authorization Request and Usage Indication messages, call function number 35 before calling 29.

NOTE:To get the Destination Protocol or the OSP support information of the destination, use function number 38 and 39 after calling 27 or 28.

NOTE:To send multiple capability indication messages, run test 101.

NOTE: To test the multiple provider functionality, run test 100. It prompts the user to enter the number of Providers that the user wants to create and then the number of transactions that need to be run for each provider.

How to Integrate the OSP Toolkit with Your Application

Your project must have access to the following:

- The compiled OSP Toolkit Library and header files,
- The compiled OpenSSL libraries (libcrypto.a and libssl.a for Unix; ssleay32.lib and libeay32.lib for Windows with the modifications stated earlier) and header files,
- (for Windows only) the pthreads library and header files,
- (if you plan to use BSAFE only) the BSAFE library (bsafe40.a for Unix; bsafe40 for Windows) and header files.

In addition to including the above libraries and files, there are two additional issues that must be resolved before including the OSP Toolkit within your application:

Initializing the Random Number Generator

Recent versions of the OpenSSL library require the initialization the Random Number Generator Engine before using certain cryptographic functionality (such as establishing SSL connection). The OSP Toolkit performs this step in the "ospopenssl.c" module using "int rand_init()" function when the toolkit is initialized. The function looks for a ".rnd" file in the current directory, and uses its content for initialization. If the file does not exist, or if it is shorter than 16 bytes, the toolkit will log a warning message; in contrast, if the file is very long, initialization may become time consuming. The current directory for "test_app" and "enroll" is "bin". Performing a default installation of the OSP Toolkit will include a "bin/.rnd" file; contents of this file will include RAND related documentation. Initialization of RAND engine is important. It is up to the vendor to re-implement the "int rand_init()" function or generate random contents for the ".rnd" file in a production environment.

Cryptographic Hardware Accelerators - (optional)

To use, you must have a supported cryptographic acceleration device, OpenSSL 0.9.8 and above (or the OpenSSL-Engine), Within the OSP Toolkit, the cryptographic hardware acceleration device is initialized when the OSPPIInit API is called with hardware acceleration set to TRUE. This occurs in the "ospopenssl.c" module using the "int cha_engine_init()" function. If your system does not have a cryptographic hardware acceleration device, if it is not supported, or if it is not configured, then pass FALSE in THE OSPPIInit API for hardware acceleration.

Appendix A: Custom Compile Options

Introduction

This section describes how to use the custom compile options to tailor the OSP Toolkit to suit your needs.

Description of Compile Options

The following table describes a list of all custom options for both Unix and NT. Some options are platform dependent—those that are not available for a particular operating system will be indicated as such. Some options take effect by simply not defining the option.

Compile Option Descriptions	Unix Option	NT Option	Production / Release	Development / Debug_Secure
TOOLKIT BUILD TYPE				
Build and install the OSP library	make install	(option not available)	X	X
Build an OSP Client Library (default)	make build	(option not available)	X	X
PERFORMANCE				
Use compiler optimization (default)	-O	(option not available)	X	
DIAGNOSTICS				
Enable full OSP debugging	OSPC_DEBUG	OSPC_DEBUG		X
Debugging symbols	-g	(option not available)		X
COMPILER				
Use Sun compiler	cc	(option not available)	X	X
Use GNU gcc (default)	gcc	(option not available)	X	X
OTHER				
Make library in destination dir	INSTALL_PATH=dir	(option not available)	X	X
Do not make library, clean only	clean	(option not available)	X	X
Use POSIX Multithreading	(option not available)	_POSIX_THREADS	X	X

Custom Compilation in Unix

The CFLAGS definition in the Makefile needs to be modified to (1) compile the toolkit in either the production or debug mode; or (2) select various options. The default build process compiles the toolkit in production mode. The CFLAG definition for debug mode is:

```
CFLAGS = -g -DOSP_ALLOW_DUP_TXN -DOSP_NO_DELETE_CHECK -
DOSP_SDK -D_POSIX_THREADS -D_REENTRANT $(GCCFLAGS) -
DOSPC_DEBUG
```

Custom Compilation in Windows

Usually, changing only one or two parameters is necessary for a custom build. In order to accomplish this, you must select a default configuration, and the flags to modify that default configuration.

Add or remove compiler options by performing the following steps:
Open the Compiler Settings dialog box:

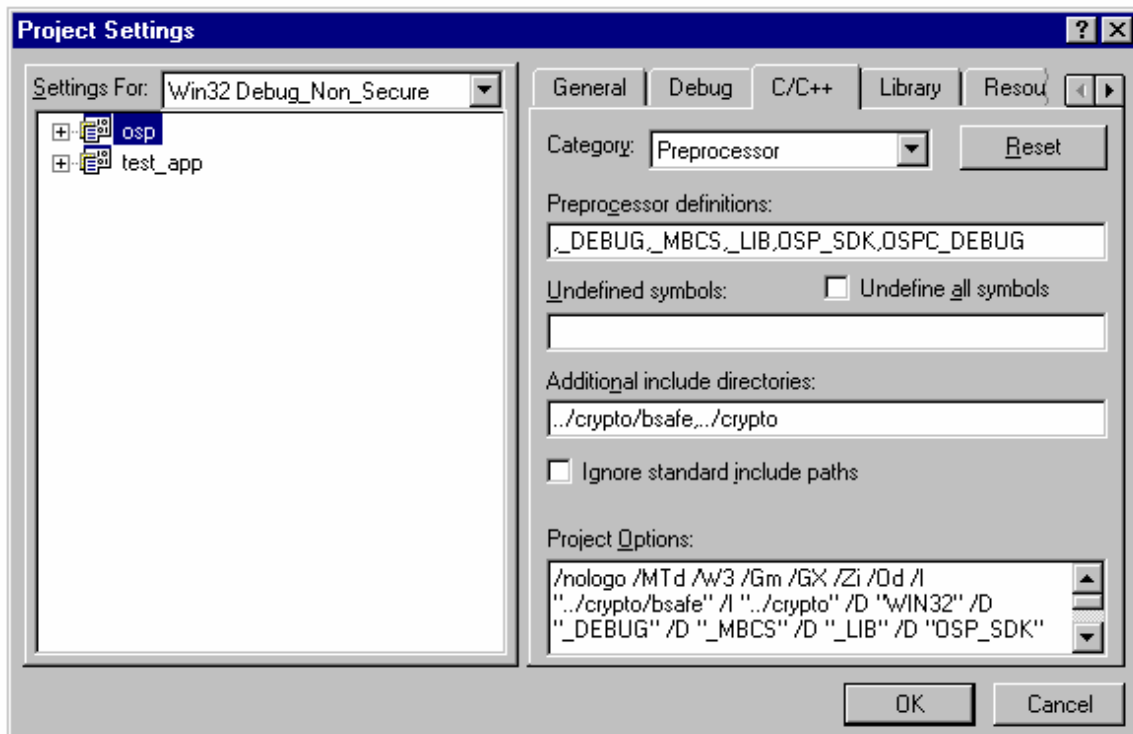
Menu: Project -> Settings ->
Tab: C/C++ -> category: Preprocessor

Select "osp" by itself (with the "Settings For:" drop down menu set to the default configuration which most closely matches the options you want in your custom build. e.g. Release, Debug_Secure).

Add or remove preprocessor definition flags by performing the following steps: In the text box "Preprocessor definitions", append or remove the desired flags to and from the existing list of flags (NOTE: the flags must be separated with commas).

WARNING: DO NOT REMOVE ANY PREPROCESSOR FLAGS NOT FOUND IN THE TABLE OF COMPILER OPTIONS IN THE TABLE ABOVE.

The following picture illustrates what the dialog box should look like after these steps have been completed:



Frequently Asked Questions

This section includes common questions about this product along with answers. If your question was not answered here, please contact support@transnexus.com with your question.

In Windows, why do I receive a fatal error stating that a file cannot be found?

If you receive an error that starts with either "fatal error C1083:" or "fatal error LNK1104:" then it means that Visual C++ cannot find at least one of your header files or library files respectively. The following table lists the most common files missing and the reason why:

File	Reason
aglobal.h	BSAFE include files are missing. Check to make sure they were placed in the correct directory.
ssl.h	OpenSSL include files are missing. Check to make sure they were placed in the correct directory.
bswift.h	Cryptographic Hardware Acceleration include files are missing. Check to make sure they were placed in the correct directory.
bsafe40.lib	BSAFE library is missing. If you are using BSAFE make sure the library is in the correct location.
libeay.lib	A least one OpenSSL library is missing. Check to make sure the libraries were placed in the correct directory.
ssleay.lib	A least one OpenSSL library is missing. Check to make sure the libraries were placed in the correct directory.
osp.lib	The OSP Toolkit library is missing. This normally occurs when the Toolkit was compiled with one option and the program is being compiled with a different option.

What are the differences between the TransNexus OSP Nexus Server and the TransNexus OSP Test Server?

There are no differences between the two. The TransNexus OSP Test Server is a TransNexus OSP Nexus Server simply configured for testing.

I need more debugging information from the OSP Toolkit. How do I get it?

The file `ospdebug.h` located within the `src` directory controls the type and amount of debug information displayed to stdout. For more information about how to configure the OSP Toolkit in this manner, please consult this file. By default, it only displays the minimal amount of debugging messages.

Why do I get the following errors when I send an Authorization Request?

```
depth=0 /CN=betabel/O=OSPServer
verify error:num=18:self signed certificate
verify return:1
depth=0 /CN=betabel/O=OSPServer
verify return:1
ERROR 14280: http response unexpected
File: ospsocket.c line: 545
expected 1XX or 2xx code: received = [http/1.1 503
```

```

server: transnexus ospserver
date: wed, 30 jan 2002 18:27:13 gmt
connection: keep-alive
keep-alive: timeout=3600, max=5000
content-length: 0
content-type: text/plain

]
num dest = 3
function return code = 0
press any key to continue...

```

The reason these errors occur has to do with self-signed certificates. OpenSSL does not like self-signed certificates, although it can be set to tolerate them. The TransNexus CA certificate which is included with the OSP Toolkit is a self-signed certificate. To get around this problem, when OpenSSL reports an error stating that the certificate is self-signed, the OSP Toolkit compensates and tries again. The key to knowing if the Authorization Request was successful is in the last three lines--as long as the function return code is zero, then the request was successful as the above example illustrates.

If the OSP Test Server will not accept SSL messages, why do we have to use OpenSSL?

The OpenSSL libraries and header files are required for successful compilation. The OSP Toolkit was designed so that it would work with either SSL or non-SSL messages--which ones depend on the URL given to the OSP Toolkit at runtime. Later, if you wish to test with an OSP Nexus Server, then the Toolkit you are using is already set for SSL messages. Also, the cryptographic algorithms are necessary to sign tokens which are required to use the TransNexus OSP Test Server.

Why does the OSP client clock need to be synchronized with the OSP Server clock?

The OSP ToolKit will only validate an OSP token if it was signed within a time window of +/- five minutes. For example, the error message below is output from test_app Transaction API function 31. The token was rejected because the time of the signed token did not match the time of the test_app.

```

Enter function number or 'q' to quit => 31
CN:0;1%0#Uosptestserver.transnexus.com10U
  OSPServer
HN:osptestserver.transnexus.com
ERROR 11130: too soon to use token
  File: osptransapi.c line: 2774
function return code = 11130

```

When compiling Enroll and Test_app in Windows, why do I get a large number of compiler errors similar to the following?

```

-----Configuration: enroll - Win32 Debug-----
Compiling...
osptnep.c
osptnepenroll.c
osptnepinit.c
osptneputil.c
Linking...
LINK : warning LNK4049: locally defined symbol "_time" imported
LINK : warning LNK4049: locally defined symbol "_abort" imported
LINK : warning LNK4049: locally defined symbol "_strncmp" imported

```

```
LINK : warning LNK4049: locally defined symbol "_sprintf" imported
LINK : warning LNK4049: locally defined symbol "_fclose" imported
LINK : warning LNK4049: locally defined symbol "_fflush" imported
LINK : warning LNK4049: locally defined symbol "_fprintf" imported
LINK : warning LNK4049: locally defined symbol "_strncpy" imported
LINK : warning LNK4049: locally defined symbol "__iob" imported
LINK : warning LNK4049: locally defined symbol "_pctype" imported
LINK : warning LNK4049: locally defined symbol "_isctype" imported
LINK : warning LNK4049: locally defined symbol "___mb_cur_max" imported
LINK : warning LNK4049: locally defined symbol "_gmtime" imported
LINK : warning LNK4049: locally defined symbol "_getenv" imported
LINK : warning LNK4049: locally defined symbol "_tolower" imported
LINK : warning LNK4049: locally defined symbol "_sscanf" imported
LINK : warning LNK4049: locally defined symbol "_strchr" imported
LINK : warning LNK4049: locally defined symbol "_signal" imported
osp.lib(ospopenssl.obj) : error LNK2001: unresolved external symbol
_SSLGetSSLstdout
s3leay32.lib(s3_lib.obj) : error LNK2001: unresolved external symbol
__imp_qsosort
libeay32.lib(stack.obj) : error LNK2001: unresolved external symbol
__imp_qsosort
libeay32.lib(a_set.obj) : error LNK2001: unresolved external symbol
__imp_qsosort
libeay32.lib(bss_file.obj) : error LNK2001: unresolved external symbol
__imp_fopen
libeay32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_fopen
libeay32.lib(conf.obj) : error LNK2001: unresolved external symbol __imp_fopen
libeay32.lib(read_pwd.obj) : error LNK2001: unresolved external symbol
__imp_fopen
libeay32.lib(bss_file.obj) : error LNK2001: unresolved external symbol
__imp_fread
libeay32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_fread
libeay32.lib(bss_file.obj) : error LNK2001: unresolved external symbol
__imp_fwrite
libeay32.lib(bss_file.obj) : error LNK2001: unresolved external symbol
__imp_setmode
libeay32.lib(bss_file.obj) : error LNK2001: unresolved external symbol
__imp_fileno
libeay32.lib(bss_file.obj) : error LNK2001: unresolved external symbol
__imp_ftell
libeay32.lib(bss_file.obj) : error LNK2001: unresolved external symbol
__imp_fseek
libeay32.lib(bss_file.obj) : error LNK2001: unresolved external symbol
__imp_fgets
libeay32.lib(b_print.obj) : error LNK2001: unresolved external symbol
__imp_vsprintf
libeay32.lib(stack.obj) : error LNK2001: unresolved external symbol
__imp_bsearch
libeay32.lib(by_dir.obj) : error LNK2001: unresolved external symbol
__imp_stat
libeay32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_DeleteDC@4
libeay32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_DeleteObject@4
libeay32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_GetBitmapBits@12
libeay32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_BitBlt@36
libeay32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_GetObjectA@12
libeay32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_SelectObject@8
libeay32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_CreateCompatibleBitmap@12
libeay32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_GetDeviceCaps@8
libeay32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_CreateCompatibleDC@4
libeay32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_CreateDCA@16
libeay32.lib(read_pwd.obj) : error LNK2001: unresolved external symbol
__imp_getch
libeay32.lib(read_pwd.obj) : error LNK2001: unresolved external symbol
__imp_fputs
libeay32.lib(read_pwd.obj) : error LNK2001: unresolved external symbol
__imp_longjmp
```

```
../bin/enroll.exe : fatal error LNK1120: 26 unresolved externals
Error executing link.exe.
```

```
enroll.exe - 33 error(s), 18 warning(s)
```

```
-----Configuration: test_app - Win32 Debug-----
Compiling...
nonblocking.c
syncque.c
test_app.c
Linking...
LINK : warning LNK4049: locally defined symbol "_fopen" imported
LINK : warning LNK4049: locally defined symbol "_fclose" imported
LINK : warning LNK4049: locally defined symbol "_fflush" imported
LINK : warning LNK4049: locally defined symbol "__setmode" imported
LINK : warning LNK4049: locally defined symbol "_fgets" imported
LINK : warning LNK4049: locally defined symbol "_fprintf" imported
LINK : warning LNK4049: locally defined symbol "_sprintf" imported
LINK : warning LNK4049: locally defined symbol "_strncpy" imported
LINK : warning LNK4049: locally defined symbol "_time" imported
LINK : warning LNK4049: locally defined symbol "_iob" imported
LINK : warning LNK4049: locally defined symbol "_abort" imported
LINK : warning LNK4049: locally defined symbol "_pctype" imported
LINK : warning LNK4049: locally defined symbol "_isctype" imported
LINK : warning LNK4049: locally defined symbol "__mb_cur_max" imported
LINK : warning LNK4049: locally defined symbol "_qsort" imported
LINK : warning LNK4049: locally defined symbol "_gmtime" imported
LINK : warning LNK4049: locally defined symbol "_getenv" imported
LINK : warning LNK4049: locally defined symbol "_tolower" imported
LINK : warning LNK4049: locally defined symbol "_strncmp" imported
LINK : warning LNK4049: locally defined symbol "_sscanf" imported
LINK : warning LNK4049: locally defined symbol "_strchr" imported
LINK : warning LNK4049: locally defined symbol "_signal" imported
osp.lib(ospopenssl.obj) : error LNK2001: unresolved external symbol
_SSLGetSSLStdout
libey32.lib(bss_file.obj) : error LNK2001: unresolved external symbol
__imp_fread
libey32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_fread
libey32.lib(bss_file.obj) : error LNK2001: unresolved external symbol
__imp_fwrite
libey32.lib(bss_file.obj) : error LNK2001: unresolved external symbol
__imp_fileno
libey32.lib(bss_file.obj) : error LNK2001: unresolved external symbol
__imp_ftell
libey32.lib(bss_file.obj) : error LNK2001: unresolved external symbol
__imp_fseek
libey32.lib(b_print.obj) : error LNK2001: unresolved external symbol
__imp_vsprintf
libey32.lib(stack.obj) : error LNK2001: unresolved external symbol
__imp_bsearch
libey32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_DeleteDC@4
libey32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_DeleteObject@4
libey32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_GetBitmapBits@12
libey32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_BitBlt@36
libey32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_GetObjectA@12
libey32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_SelectObject@8
libey32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_CreateCompatibleBitmap@12
libey32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_GetDeviceCaps@8
libey32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_CreateCompatibleDC@4
libey32.lib(md_rand.obj) : error LNK2001: unresolved external symbol
__imp_CreateDCA@16
libey32.lib(by_dir.obj) : error LNK2001: unresolved external symbol
__imp_stat
libey32.lib(read_pwd.obj) : error LNK2001: unresolved external symbol
__imp_getch
```

```

libeay32.lib(read_pwd.obj) : error LNK2001: unresolved external symbol
__imp__fputs
libeay32.lib(read_pwd.obj) : error LNK2001: unresolved external symbol
__imp__longjmp
../bin/test_app.exe : fatal error LNK1120: 22 unresolved externals
Error executing link.exe.

test_app.exe - 24 error(s), 22 warning(s)

```

Usually, these errors are caused by an improper compilation of OpenSSL. Delete everything in the temp32 directories, and start again at step 4 of the "Building the Toolkit in Windows" section again.

Sometimes these errors may also come if while configuring the test_app and enroll for pthreads, you check the "Ignore all default libraries" box. Your project settings should look like as shown in diagram 2 of Appendix A. Make sure that you do not check the "Ignore all default libraries" box.

I received a certificate, but there are errors. Did I successfully enroll?

```

output buffer after operation: operation=request
output buffer after nonce: operation=request&nonce=2604418792822824
ERROR 16030: X509 CertInfo context is null pointer
File: ospx509.c line: 634
ERROR 12080: Unable to get Local Certificate
File: ospopenssl.c line: 491
depth=0 /CN=ospptestserver.transnexus.com/O=OSPServer
verify error:num=18:self signed certificate
verify return:1
depth=0 /CN=ospptestserver.transnexus.com/O=OSPServer
verify return:1
The certificate request was successful.
MIIBTjunAmA0EQCm3c9pwWYjHIowF0/Ue92TMAbG9SgSIB3DQEBAUAMdsxJTAj
BgN7BAMTHG2zcHRlc3RzZXJ2ZXIudHJhbnYuZXhhcy5jb20xEjAQBGNVBAoTCU9T
UFNlcnZlcjAeFw0wMjAyMDQxOTQyMjdaFw0wMzAyMDUxOTQyMjdaMIGAMiswCQYD
VQQGEWJzUz5LMAwwA1UECBMCR0ExDDAKBgNVBACTA0FsTDENMAsgA1UEFhMESklN
RDEPML0GA1UECzMGVGVzdGVyMQwwCgYDVkQDEwNkaW0xKDAmBgkqhkiG9w0BCQEW
GWppbS5kYWw0b25AdHJhbnNuZhh1cy5jb20wXDANBgkqhkiG9w0BAQEFAANLADBI
AkEAp3fkMb10pzruMuNaae4FOAq5UJlkwbr13a+lFYH2H7s9ima/zQq6dCOZGTi
6LJEhm/2iFKyXs3dYkgd7uXNAQIDAQAABMA0GCSqGSIB3DQEBAUAA0EAMcmeO+Gu
YGF8EKZ6w+Wp+2zOWyiLeQof0fFVxUUF9PFJiaUmT6+h9VPu4LSORbELP/fe8Tno
gdwMcsNRyFP02A==
Press any key to continue.

```

As long as you received a certificate, you successfully enrolled; the error can be ignored. The example above illustrates a successful enrollment.

While compiling the OSP Toolkit in Windows, I received approximately 94 errors and 65 warnings all in ospcryptowrap.c and in ospopenssl.c. Why does this happen?

Most likely, the openssl include files you copied into the crypto directory are all empty (zero length). Try copying them again from the inc32 directory within the OpenSSL directory. Try compiling the OSP Toolkit again.

Why do I get the following error while trying to obtain a certificate request and private key (when enrolling)?

```

Using configuration from /usr/local/ssl/openssl.cnf
Unable to load config info
Loading 'screen' into random state - done

```

```
Generating a 512 bit RSA private key
.....+++++
.....+++++
writing new private key to 'pkey.b64'
-----
unable to find 'distinguished_name' in config
problems making Certificate Request
```

OpenSSL cannot find its configuration file. Find the "openssl.cnf" file (on Windows, it might be just "openssl"); it is usually located in the "apps" directory within the OpenSSL directory. When this file is found, follow the instructions regarding appending text to the first OpenSSL command..

I am trying to connect to the OSP Server, but the request just blocks holding up all my resources. What is wrong?

Make sure that the server address provided in the configuration file, *test.cfg*, is correct. If the server is unavailable at the address specified in the file, the TCP *Connect* request will not return back till it times out. This time out value by default is different across different platforms. For some it is small and for some, relatively large. If you are running on a Solaris box, it is quite possible that you experience a reasonably long hold up period. This begets from the fact that the default TCP connection retry time out value is 180 seconds. This connection time out of 3 minutes might be a delay too long in certain cases. If you want to improve on this, change the TCP parameters to have a lower time out. The three variables governing the retransmission procedure are: *tcp_ip_abort_cinterval*, *tcp_rexmit_interval_max* and *tcp_rexmit_interval_initial*. Their default values on Solaris are 180000, 60000 and 3000 ms respectively. These values can be obtained using the "*ndd -get /dev/tcp/variable*" command. In order to change these values, "*ndd -set /dev/tcp/variable <value>*" can be used. However, it should be kept in mind that this solution affects all the TCP implementations on the box. Also, realize that the Toolkit's response time-out does not control connection time out. These are two different parameters.

Why will my application not establish SSL connection?

Make sure that the current directory includes ".rnd" file and it is at least 16 bytes long.

What hardware devices are supported?

For the most up-to-date list, refer to the online OpenSSL documentation at www.openssl.org/docs. The OSP Toolkit has been tested with OpenSSL-Engine-0.9.6c which supports "CryptoSwift", "nCipher", "Atalla", "Aep", "SureWare", "UBSEC", and "CryptoApps".

I don't have a cryptographic hardware acceleration device, what do I do?

You don't have to have one. Set the hardware initialization parameter in the *OSPPIInit* function to FALSE. This will tell the OSP library that you do not hardware support.

My version of OpenSSL does not have Engine API, and I get compile time errors. What do I do?

This is not a problem. Simply compile the toolkit without cryptographic hardware acceleration support.

How do I know if my cryptographic hardware acceleration device is being used?

Refer to your vendor's documentation. For example, Cryptoswfit includes "csdiag" utility which displays usage information for the acceleration card.

I have a cryptographic hardware acceleration device but it is NOT being used, why?

Make sure that you are setting the hardware support parameter in the toolkit API to TRUE and that you don't get a warning message when the application is started.

I have more than one supported cryptographic hardware acceleration device how do I know which one will be used?

The OSP Toolkit will iterate through the list of all supported devices and will select the first one found. If you want to use a specific device, re-implement the function "int cha_engine_init()" (in the "ospopenssl.c" module).

What can a cryptographic hardware acceleration device be used for?

Cryptographic hardware acceleration devices may be used for cryptographic operations required for Secure Sockets Layer (SSL) and token validation. Also, its uses are determined on the crypto services implemented by the device.

Why do I get errors saying 'undefined symbols' when I try to build the toolkit?

You will get these errors if you happen to reverse the order of the openssl libraries to be included. The correct order is – libcrypto, libssl. Look into the make file and make sure that it is the same. Reversing the order will generate these errors.

What values should I configure for - HttpRetryLimit, HttpRetryDelay and HttpRetryTimeOut ?

The HttpRetryDelay parameter should ideally be 0, which means no lag between retrying connection attempts to the server. The HttpRetryTimeOut parameter can be anything in the range of 3-5 seconds. The HttpRetryLimit should be, in some way related to the number of Service Points configured. An inverse relation between the two only seems logical. If there are quite a few SP's configured, you would like to keep the HttpRetryLimit value low so that the toolkit gets a chance to ping each server on the list within a reasonable period of time. For e.g., if there are 5 SP's configured with time out value of 3 seconds, and the toolkit retries each SP 5 times before trying the next one in list, it would take $5 * (5+1) * 3 = 90$ seconds before all the SP's can be tried. This may be a value too high in certain cases.

These values should be set keeping in mind the real time nature of the authorization request. They should be set optimally to have higher throughput both within the toolkit and over the network. Small timeout values would mean overloading the toolkit connection manager and high values would compromise the real time nature of the message. The UsageIndication response may sometimes time out. We advise the application to call UsageIndication again upon timeout, rather than increase the retry limit or the retry timeout parameters in the toolkit.

What encryption mechanisms are supported in the toolkit ?

The toolkit supports all the SSL/TLS ciphersuites. It does not do any S/MIME signatures and for tokens, it does RSA, MD5, and DES.