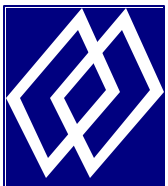


TransNexus

OSP Toolkit

How to Build and Test the OSP Toolkit

Release 2.5.5

5 February 2002

OSP Toolkit

How to Build the Toolkit

Release 2.5.5

5 February 2002

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 1

OSP Toolkit Primer 1

Building the Toolkit in Unix 2

- Introduction 2
- Unpacking the Toolkit 3
- Preparing to Build the OSP Toolkit 4
- Building the OSP Toolkit 4
- Building the Enrollment Utility 5
- Building the Test_App Client Simulator 5

Building the Toolkit in Windows NT 6

- Introduction 6
- Prepare Third Party Software and Unpack the OSP Toolkit Distribution 6
- Configure the OSP Toolkit Distribution for Windows NT 8
- Compile the OSP Toolkit for Windows NT 9
- Compile the Enrollment Utility for Windows NT 10
- Compile the Test_App Utility for Windows NT 10

Testing the Compiled Toolkit in Unix and NT 11

- Overview 11
- Preparations for Testing 11
- Using the enroll program on Unix and Windows NT 12
- Running the Client Simulator (test_app): 17

Appendix A: Configuring the OSP Toolkit for pthreads in Windows NT 22

- Introduction 22
- Obtaining a Pthreads Library and Supporting Files for Windows NT 22
- Configuring the OSP Toolkit , Test_App Client Simulator, and Enroll Preprocessor Options 23
- Configuring the Test_App Client Simulator and Enroll for the Pthreads Library 24

Appendix B: Custom Compile Options 25

- Introduction 25
- Description of Compile Options 25

Custom Compilation in Unix..... 26
Custom Compilation in NT..... 27
Frequently Asked Questions 28

Introduction

This document provides build instructions for of release 2.5.5 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 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 reader is advised to read the *OSP Toolkit Introduction* and *Implementation Guide* for an overview of OSP before using this document. The OSP Toolkit has been tested with the following operating systems and compilers: Sun Solaris 2.6/2.7 (using GCC and SparcWorks C++ 5.0), Microsoft Windows NT 4.0 (using Visual C++ 6.0).

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.

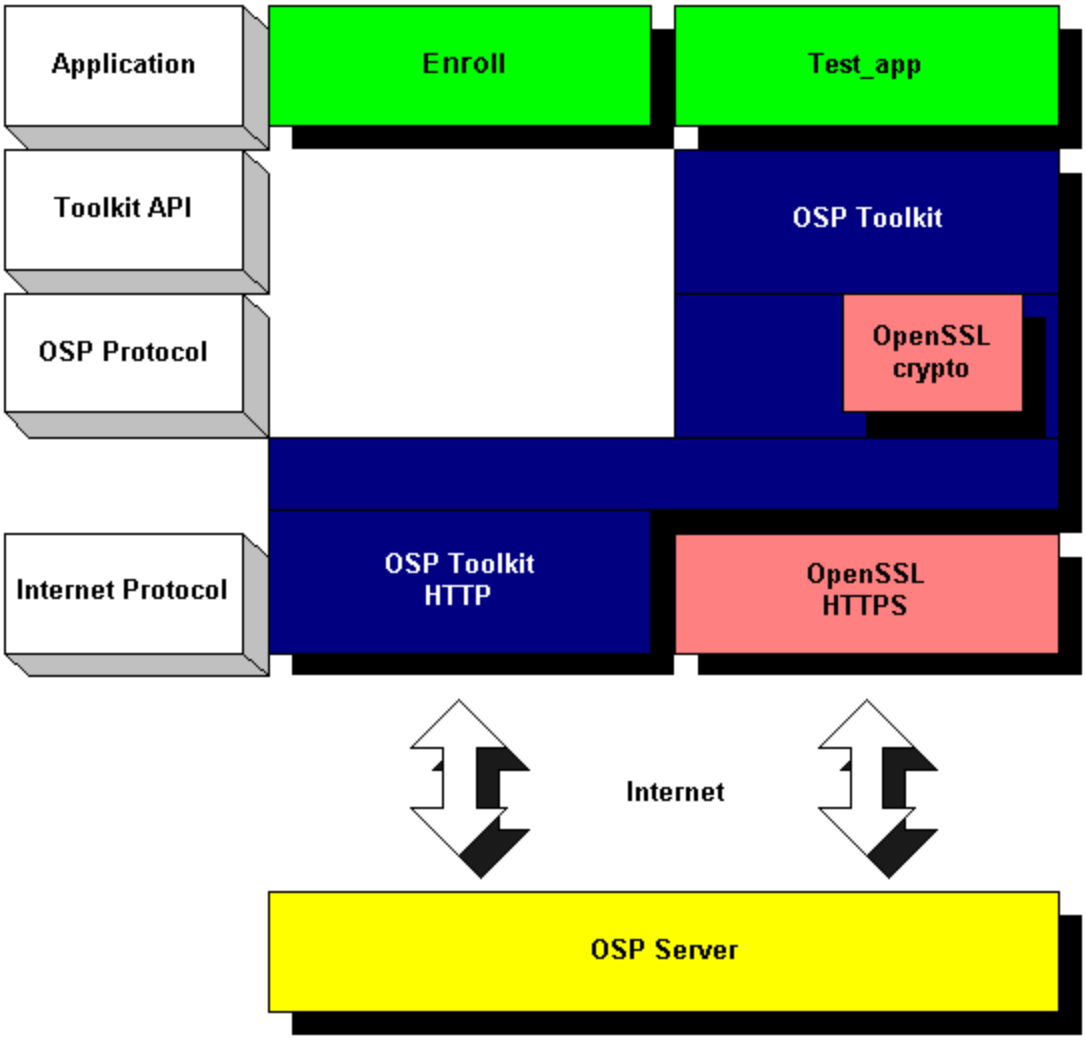


Figure 1: Structure and Layers of the OSP Toolkit

Building the Toolkit in Unix

Introduction

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 0.9.4 recommended) from www.openssl.org
- 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.

- 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
```

New directories will be created within the same directory as the tar file. Within this directory, you will find directories and files similar to what is listed below (if the command "ls -F" is executed):

```
> ls -F
OSPToolkit-2_5_4.tar              enroll/
OSPToolkit-2_5_4-File_List.txt    include/
OSPToolkit-2_5_4-RelNotes.txt     lib/
README.txt                        license.txt
bin/                              src/
crypto/                           test/
doc/
```

Preparing to Build the OSP Toolkit

4) 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/.

5) 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.

6) 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

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

```
cd src; chmod a+x osp_sdk_compile.ksh

make clean; osp_sdk_compile.ksh
```

The following menu will be displayed:

```
        OSP Build Menu
-----------------------------
1. Client Production Library
2. Client Development Library
3. Client Non-Secure Library

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

The following table identifies which default features are activated with each compile option:

Default Feature	Production	Development	Non-Secure
Tokens Signed	Yes	Yes	No
Uses Hardware Acceleration*	Yes	No	No
Debug Information Displayed	No	Yes	Yes
* Contact TransNexus for more information about cryptographic hardware acceleration. See Appendix B for more information and/or custom compile options.			

Enter the compilation option and hit "Enter". The "Development" option is recommended for a first time build. If you need to customize your build, see Appendix B for details of how to accomplish this. After the script has finished running, select "q" to exit the script. You may see a number of warnings relating to OpenSSL--this may be ignored. Compilation is successful if there are no errors anywhere in the compiler output.

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".

8) 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.

Building the Test_App Client Simulator

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

9) 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.

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

Building the Toolkit in Windows NT

Introduction

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

- OpenSSL (required) - Open Source SSL protocol and Cryptographic Algorithms (version 0.9.4 recommended) from www.openssl.org
- 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.
- Pthreads Library (strongly recommended) - 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
- C compiler (required) - Any C compiler should work. All the examples in this document are based on Visual C++ 6.0. For details purchasing this compiler, 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.

Even if you have OpenSSL installed on your computer, you MUST download this version and make the following modifications for the NT version of the OSP Toolkit to work.

2) Download the OpenSSL 0.9.4 distribution from www.openssl.org.

3) Download and install Perl for Win32 in order to compile OpenSSL (available from http://www.activestate.com/ActivePerl

4) Compile OpenSSL 0.9.4 for Visual C++ (most of this material is a modified version of the Install.w32 file within the OpenSSL directory).

   - Run Configure by executing the following commands at the DOS prompt:

```
cd <OpenSSL_directory_path>

perl Configure VC-WIN32
```

where <OpenSSL_directory_path> is the path to the OpenSSL directory.

- Build the makefiles by executing the following command at the DOS prompt:

```
ms\do_ms
```

- Modify the file openssl-0.9.4\ms\ssleay32.def with any text editor:

  Add the following line of code between ERR_load_SSL_strings and SSL_CIPHER_description (line 17):

```
SSLGetSSLstdout
```

- Modify the file openssl-0.9.4\ssl\ssl_lib.c with any text editor:

  Add the following lines of code at the end of the file (which will become lines 1949 thorugh 1953):

```
FILE *SSLGetSSLstdout (void)
{
    return (stdout);
}
```

- Modify the file openssl-0.9.4\ssl\ssl.h with any text editor:

  Add the following line of code (line 1009):

```
FILE *SSLGetSSLstdout(void);
```

just after this line of code on line 1008:

```
long SSL_CTX_ctrl(SSL_CTX *ctx, int cmd, long larg, char *parg);
```

- Compile OpenSSL by executing the following command at the DOS prompt:

```
nmake -f ms\ntdll.mak
```

NOTE: If you have to do this step again, first delete all files within the tmp32out directory contained within the OpenSSL directory.

- Verify compilation by running the OpenSSL regression tests by executing the following commands at the DOS prompt:

```
cd out32dll

..\ms\test
```

If there are no errors reported, then the tests and the compilation were successful.

- Be sure to permanently set the OpenSSL path (to the directory out32 or out32dll --whichever has the openssl.exe program) on this box. This is accomplished by going to the system control panel (within the control panels directory) and selecting "Environment". Add the openssl path to the PATH variable.

- Using any text editor (WordPad, NotePad, edit, etc.), edit the file "openssl.cnf" contained within the "apps" directory within the OpenSSL directory. (it might be simply "openssl" on some Windows versions). At the top of the file is the following text:

```
#
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
#

RANDFILE           = $ENV::HOME/.rnd
oid_file           = $ENV::HOME/.oid
oid_section        = new_oids
```

Remove the following text: "`$ENV::HOME/`" in both places, and save the file (remove the slash, but keep the period).

5) Copy the gzipped file to the desired NT directory.

6) Unzip the tar file by using any application that unzips and untars files. The unzipped files may be placed in any desired directory.

Configure the OSP Toolkit Distribution for Windows NT

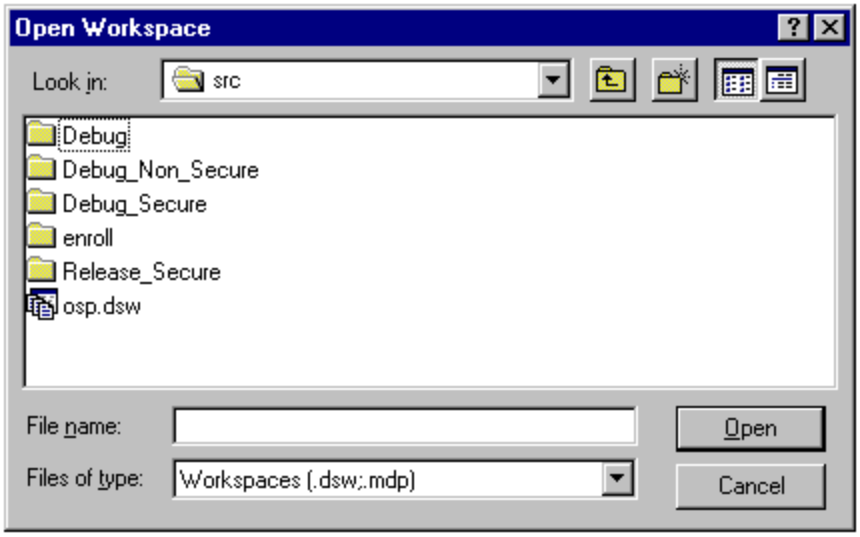
7) Copy the OpenSSL header files (ones with the *.h extension) into the crypto\openssl directory within the Toolkit. The OpenSSL header files are located under the inc32\openssl directory (NOT from the include\openssl directory).

8) Copy the OpenSSL library files ssleay32.lib and libeay32.lib from the openssl out32dll directory into the lib directory within the Toolkit. Also, copy the dynamically link libraries ssleay32.dll and libeay32.dll from the openssl out32dll directory into the bin directory within the Toolkit.

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

Menu: File -> OpenWorkspace

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



11) If you want to use pthreads instead of the Windows native threads, follow the instructions in Appendix B. **We strongly recommend that pthreads be used instead of Windows native threads.**

Compile the OSP Toolkit for Windows NT

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

12) 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_Secure	Debug_Non_Secure
Tokens Signed	Yes	Yes	No
Uses Hardware Acceleration*	Yes	No	No
Debug Information Displayed	No	Yes	Yes
* Contact TransNexus for more information about cryptographic hardware acceleration. See Appendix B for more information and/or custom compile options.			

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

Compile the Enrollment Utility for Windows NT

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

Project -> Set Active Project -> enroll

15) 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
Debug_Non_Secure	Debug

16) Compile the Enroll utility.

Menu: Build -> Build Enroll.exe

Compile the Test_App Utility for Windows NT

Build the test_app utility, and run the program:

17) Set the active project to "test_app"

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

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

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

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

20) Compile test_app.

Menu: Build -> Build Test_app.exe

Testing the Compiled Toolkit in Unix and NT

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 "#". By default, the test_app configuration file is set so the OSP TransNexus Test Server returns two destination IP addresses, 1.1.1.1 and 2.2.2.2 in random order.

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_00001.dat
Local Certificate	localcert_00001.dat
CA Certificate	cert_00001.dat

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 program on Unix and Windows NT

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.

Go to the bin directory within the OSP Toolkit directory.

```
cd bin
```

1) Generate a public and private key pair as well as a certificate request.

Execute the following command at the prompt:

```
openssl req -outform PEM -nodes -newkey rsa:512 -md5 -new
      -out certreq.b64 -keyform PEM -keyout pkey.b64
```

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

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

where <openssl_path> is the location of your OpenSSL directory. This is the same openssl.cnf file you edited earlier for Windows NT.

NOTE: On Windows NT, 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.b64" and the private key "pkey.b64". The public key is contained in the certificate request.

2) Isolate the encoded certificates by manually editing the files:

A) Edit the certreq.b64 file within a text editor (e.g. vi, emacs, WordPad, NotePad, etc.).

B) Within certreq.b64, remove everything above and including the text

```
-----BEGIN CERTIFICATE REQUEST-----
```

and everything below and including the text

```
-----END CERTIFICATE REQUEST-----
```

Also, remove all newline characters (in other words, put all the text on one line).

C) Save the changes using the same filename (replacing the original file).

D) Edit the pkey.b64 file within a text editor (e.g. vi , emacs, WordPad, NotePad, etc.).

E) Within pkey.b64, remove everything above and including the text

```
-----BEGIN RSA PRIVATE KEY-----
```

and everything below and including the text

```
-----END RSA PRIVATE KEY-----
```

Also, remove all newline characters (in other words, put all the text on one line).

F) Save the changes using the same filename (replacing the original).

Check to make sure the certificates are in the proper format. The contents of the files should be comprised only of the characters from "a" through "z", "A" through "Z", "0" through "9", "+", and "/" (sometimes with "=" at the end of the file) all on on line. For example:

```
MIIBy9b68yjCAuG2aNrPkVSpBnTRQvcRS9yGdS7/G1v+hhjBP1DVfAXl4XwbJ
xYkMCyIKtgwIDAQABmVBAoTCU9TUFNlcnZlcjAeFw0wMDEyMDQxMzU5MThaFw
0wMTEyMDUxMzU5MThaMeiC9XIgZ+17/WzwDQYJKoZIhvc+AQE/BQAwJzERMA8
GA1UEAxMId2lsbGlhbXMxEjAQBgN70GCSqFSIf3DQEBAQUAt4GNAeCKiQKBgQ
C40wqtMdP4MrQrrCLNan2xE4NV0t49w58m4EuQMm1C3Fjnd1BilmHnJYWdsKm
oBpAGf9o402/vLGrSePwftx3bFZ94aQKLcA15OtIUhwKhYrjwiYsmk85YXHD/
8Vl6RjOxGQxCzAJBgNVBAYTAlVtRAwDgYDooQIEwdHZW9yZ2lhqRAwDgHDV2Q
sEwdBdGxhbnRhMqlwHwYDVQQKExhJbnRlcm5ldCBXaWR2EwZdqQcTuZaJV/Bn
aXRzIFB0eSBcdGQxDjAMBgNVBvMTBiRldjAxMIofMA0GPSqGrIb3DQEPBAUAA
0EAvvImOBao8PjgYSVmsvRy00bJoGHI7znEqEC2+PLoUIH6PzNEJlzYqs7s8r
6Bl==
```

Do this step again if unsuccessful.

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

Execute the following command at the prompt:

```
enroll -function getcacert -caurl http://osptestserver.transnexus.com:80/tep
```

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, copy the resulting certificate into a new file with the filename cacert.b64. From this file, isolate the resulting encoded certificate by manually editing the files:

A) Edit the cacert.b64 file within a text editor (e.g. vi, emacs, WordPad, NotePad, etc.).

B) Within cacert.b64, remove everything above and including the text

```
CA certificate received
```

and everything below and including the text

```
Press any key to continue.
```

Also, remove all newline characters (in other words, put all the text on one line).

C) Save the changes using the same filename (replacing the original file).

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 "/" all on on line (see the earlier example). Do this step again if unsuccessful.

4a) Enroll the device and receive a local certificate using UNIX.

For Unix, execute the following command at the prompt:

```
enroll -function request -username trans -password nexus
    -customer 1000 -device 1000 -cacert `cat cacert.b64`
    -certreq `cat certreq.b64` -sslurl
    https://osptestserver.transnexus.com:443/tep
```

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

NOTE: The ` is a backward quote. The command will not work otherwise.

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

```
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
```

If there are no errors, copy the resulting certificate into a file and save it as localcert.b64. With this file, isolate the resulting encoded certificate by manually editing the files:

1) Edit the localcert.b64 file within a text editor (e.g. vi, emacs, NotePad, WordPad, etc.).

2) Within localcert.b64, remove everything above and including the text

   ```
   The certificate request was successful.
   ```

   and everything below and including the text

   ```
   Press any key to continue.
   ```

   Also, remove all newline characters (in other words, put all the text on one line).

3) Save the changes using the same filename (replacing the original).

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 "/" all on on line. Do this step again if unsuccessful.

4b) Enroll the device and receive a local certificate using NT.

For NT, create a new text file (in NotePad, WordPad, etc.) and copy this command into this file (the command all on one line). Replace "`cat cacert.b64`" with the CONTENTS of the cacert.b64 file, and replace "`cat certreq.b64`" with the CONTENTS of the certreq.b64 file. Save this file as "enroll.bat"; this becomes a DOS batch file. For example, the file might look like the following:

```
enroll -function request -username trans -password nexus
-customer 1000 -device 1000 -cacert MIIBy9b68yjCAuG2aNrP
kVSpBnTRQvcRS9yGdS7/G1+hhjBP1DVfAXl4XwbJxYkMCyIKtgwIDAQA
BmVBAoTCU9TUFNlcnZlcjAeFw0wMDEyMDQxMzU5MThaFw0wMTEyMDUxM
zU5MThaMeiC9XIgZ+17/WzwDQYJKoZIhvc+AQE/BQAwJzERMA8GA1UEA
xMId2lsbGlhbXMxEjAQBgN70GCSqFSIf3DQEBAQUAt4GNAeCKiQKBgQC
40wqtMdP4MrQrrCLNan2xE4NV0t49w58m4EuQMm1C3Fjnd1BilmHnJYW
dsKmoBpAGf9o402/vLGrSePwftx3bFZ94aQKLcA15OtIUhwKhYrjwiYs
mk85YXHD/8Vl6RjOxGQxCzAJBgNVBAYTAlVtRAwDgYDooQIEwdHZW9yZ
2lhqRAwDgHDV2QsEwdBdGxhbnRhMqlwHwYDVQQKExhJbnRlcm5ldCBXa
-certreq Nan2xE4NV0t49w58m4EuQMm1C3Fjnd1BilmHnJYWdsKmoBp
AGf9o402/vLGrSePwftx3bFZ94aQKLcA15OtIUhwKhYrjwiYsmk85YXH
D/8Vl6RjOxGQxCzAJBgNVBAYTAlVtRAwDgYDooQIEwdHZW9yZ2lhqRAw
DgHDV2QsEwdBdGxhbnRhMqlwHwYDVQQKExhJbnRlcm5ldCBXaWR2EwZd
qQcTuZaJV/BnaXRzIFB0eSBcdGQxDjAMBgNVBvMTBiRldjAxMIofMA0G
PSqGrIb3DQEPBAUAA0EAvvImOBao8PjgYSVmsvRy00bJoGHI7znEqEC2
+PLoUIH6PzNEJlzYqs7s8r6Bl== -sslurl https://osptestserve
r.transnexus.com:443/tep
```

Run it by typing in the name of the batch file at the prompt.

If there are no errors, copy the resulting certificate into a file and save it as localcert.b64. With this file, isolate the resulting encoded certificate by manually editing the files:

3) Edit the localcert.b64 file within a text editor (e.g. vi, emacs, NotePad, WordPad, etc.).

4) Within localcert.b64, remove everything above and including the text

   ```
   The certificate request was successful.
   ```

   and everything below and including the text

   ```
   Press any key to continue.
   ```

   Also, remove all newline characters (in other words, put all the text on one line).

3) Save the changes using the same filename (replacing the original).

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 "/" all on on line. Do this step again if unsuccessful.

5) Convert the files from base64 to binary

Execute the following commands at the command prompt:

```
openssl enc -base64 -in cacert.b64 -out cert_00001.dat -d -a -A

openssl enc -base64 -in pkey.b64 -out pkey_00001.dat -d -a -A

openssl enc -base64 -in localcert.b64 -out localcert_00001.dat -d -a -A
```

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_00001.dat
Local Certificate	localcert_00001.dat
CA Certificate	cert_00001.dat

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) GetAuthorityCertificates           4) SetAuthorityCertificates
 5) GetHTTPMaxConnections              6) SetHTTPMaxConnections
 7) GetHTTPPersistence                 8) SetHTTPPersistence
 9) GetHTTPRetryDelay                 10) SetHTTPRetryDelay
11) GetHTTPTimeout                    12) SetHTTPTimeout
13) GetLocalKeys                      14) SetLocalKeys
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) RequestReauthorisation
31) ValidateAuthorisation               32) ReportUsage
33) TransactionInitializeAtDevice(OGW)34) TransactionInitialize(TGW)
35) TransactionReinitializeAtDevice   36) TransactionRecordFailure
-------------------------------------------------------------------
Miscellaneous Tests
-------------------------------------------------------------------
37) Verify XMLTEST Signature          38) Cert Create Delete
39) Key Create Delete                 40) Show Cert Chain
41) 500    Test Calls                 42) Show Version
99) Sleep for 2 seconds
-------------------------------------------------------------------
Performance tests
-------------------------------------------------------------------
100) 1000 Full calls
-------------------------------------------------------------------
Enter function number or 'q' to quit =>
```

Use the following sequence of function numbers to test OSP ToolKit API functions used 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.

```
1, 100, 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.

Appendix A: Configuring the OSP Toolkit for pthreads in Windows NT

Introduction

Originally, the OSP Toolkit was designed to do multithreading using pthreads. An equivalent translation was ported to Windows NT; however, an anomaly was found using native Windows NT threads during intense performance tests. When a pthreads library for Windows was compiled and used with the OSP Toolkit, the anomaly disappeared. As a result, we now recommend the OSP Toolkit be compiled with a pthread library on Windows NT until we find the cause of the anomaly.

Obtaining a Pthreads Library and Supporting Files for Windows NT

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

```
mkdir pthreads
```

2) Create two other directories within the new directory:

```
cd pthreads

mkdir include

mkdir lib

cd ..\..
```

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

ftp://sources.redhat.com/pub/pthreads-win32/dll-latest

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

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

pthreadVC.lib (place this in the "pthreads\lib" directory within the Toolkit)

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

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

Configuring the OSP Toolkit , Test_App Client Simulator, and Enroll Preprocessor Options

6) Open the compiler settings dialog box:
Menu: Project -> Settings

7) In the upper left hand corner is a drop down menu titled "Settings for:" Select "All Configurations"

8) Select "osp", "test_app", and "enroll". This can be accomplished by first selecting "osp" by using the left mouse button, then while holding the control key, select "test_app", then "enroll".

9) Add the pthreads header files to the header file list by performing the following steps:

Tab: c/c++ -> category: Preprocessor

In the text box, "Additional Include Directories", append the following path of the pthread include directory (NOTE: precede this text with a comma):

..\pthread\include

Set the pthread preprocessor definition flag by performing the following steps: In the text box "Preprocessor definitions", append the following flag to the existing list of flags (NOTE: precede this text with a comma):

_POSIX_THREADS

The following picture illustrates what the dialog box should look like after steps 6 through 9 are completed:

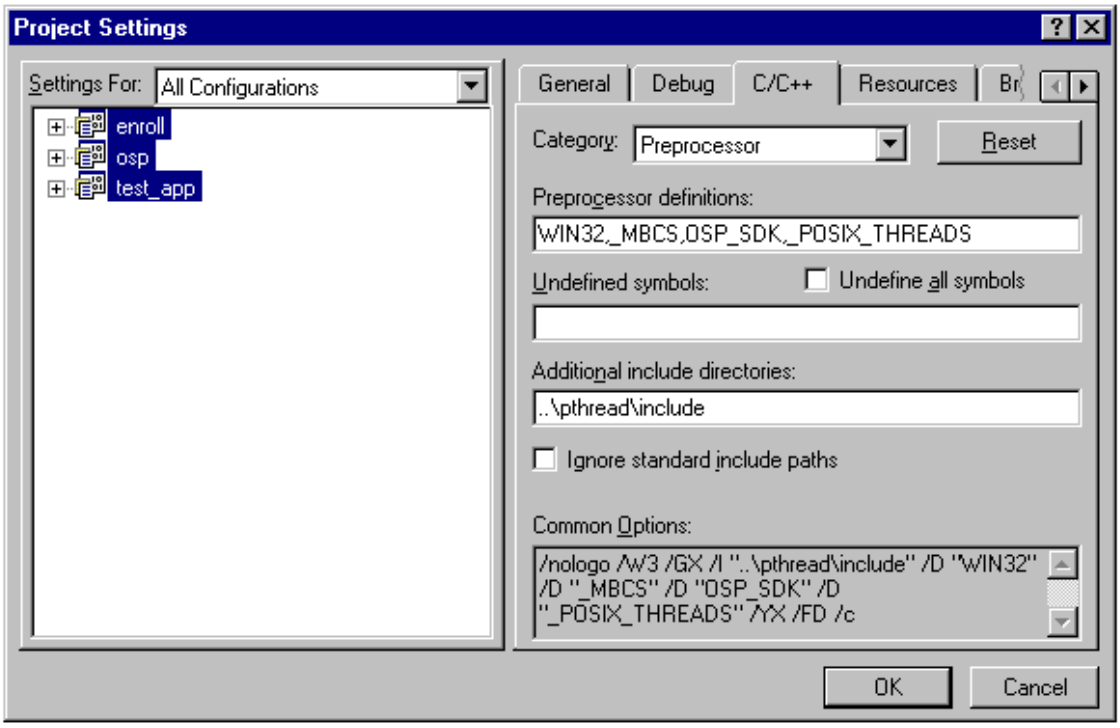


Diagram 1: Project Settings dialog box illustrating the pthreads preprocessor configurations for "osp", "test_app", and "enroll".

Configuring the Test_App Client Simulator and Enroll for the Pthreads Library

10) Select "test_app" and "enroll" (with the "Settings For:" drop down menu set to "All Configurations").

11) Add the pthreads library by performing the following steps:
With the "Settings" dialog box still on the screen:

Tab: Link -> category: Input

In the text box, "Additional Library Path", append the following path of the pthread library directory (NOTE: precede this text with a comma):

..\pthread\lib

21) Add the library module name to the list of libraries

In the text box, "Object/library modules", append the following name of the pthread library (NOTE: do NOT add commas here):

pthreadVC.lib

The following picture illustrates what the dialog box should look like after steps 10 through 12 are completed:

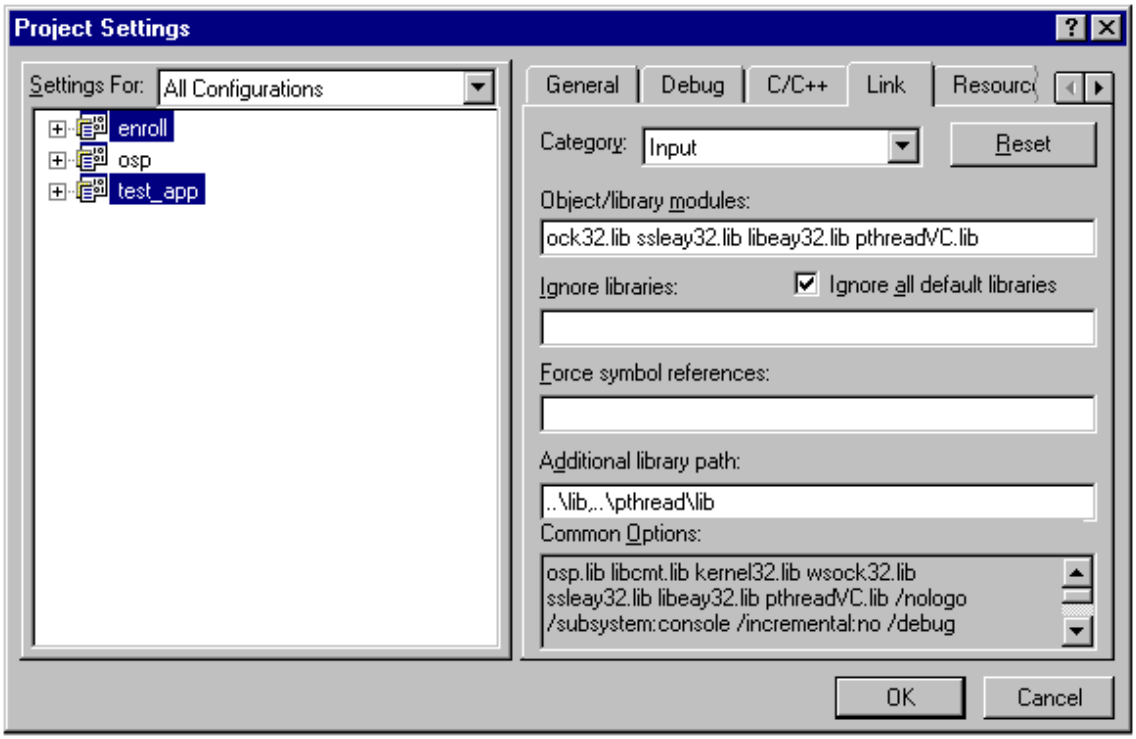


Diagram 2: Project Settings dialog box illustrating the pthreads link configurations for "test_app" and "enroll".

At this point, the OSP Toolkit and supporting software should be configured to use pthreads instead of Windows threads.

Appendix B: 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	Non-Secure / Debug_Non_Secure
TOOLKIT BUILD TYPE					
Build an OSP Client Library (default)	client	(option not available)			
TOKEN SECURITY					
Issue signed <TokenInfo> tokens (default)	secure_token	OSPC_USE_SIGNED_TOKEN	**X**	**X**	
Issue clear text <TokenInfo> tokens	non_secure_token	(do not define above option)			**X**
S/MIME MESSAGE SUPPORT					
Produce S/MIME Requests and Verify Responses	smime	ENABLESMIME			
SSL CLIENT AUTHENTICATION					
Enable SSL Client Authentication	ssl_client_auth	OSPC_ENABLE_SSL_CLIENT_AUTHENTICATION			
CRYPTOGRAPHIC LIBRARY					
Use BSAFE	use_bsafe	_BSAFE			
Use OpenSSL (default)	use_openssl	(do not define above option)	**X**	**X**	**X**
TOKEN ENCODING					
CDATA encode tokens	cdata_only	OSPC_USE_CDATA_ONLY			**X**
Base64 encode tokens (default)	b64_encode	(do not define above option)	**X**	**X**	
PERFORMANCE					
Use -O3 level compiler optimization (default)	optimized	(option not available)	**X**		
Enable hardware crypto acceleration (default)	hardware_accel	OSPC_HWE_CRYPTOSWIFT	**X**		
Disable hardware crypto acceleration	no_hardware_accel	(do not define above option)		**X**	**X**

Compile Option Descriptions	Unix Option	NT Option	Production / Release	Development / Debug_Secure	Non-Secure / Debug_Non_Secure
DIAGNOSTICS					
Enable full OSP debugging	debug	OSPC_DEBUG		X	X
Use -g (w/gcc) compiler debug option	with_symbols	(option not available)		X	X
Enable gprof profiling	gprof	(option not available)			
COMPILER					
Use SparcWorks compiler	spro	(option not available)			
Use GNU gcc (default)	gcc	(option not available)	X	X	X
Use GNU g++	g++	(option not available)			
OTHER					
Make library in destination dir	dest_dir=dir	(option not available)			
Do not make library, clean only	clean_only	(option not available)			
Use POSIX Multithreading	(option not available)	_POSIX_THREADS			

Custom Compilation in Unix

Usually, changing only one or two parameters are necessary for a custom build. In order to accomplish this, you must select a default configuration, and the flags to modify that default configuration. Use the compilation script command in the following form:

```
./osp_sdk_compile.ksh <default_opt> client -o <o1> -o <o2>
```

where <default_opt> is one of the default options in the table below, and <o1> and <o2> are the names of the compile options in the above table.

Default Configuration	<default_option>
Production	-p
Development	-d
Non-Secure	-n

For example, if you wanted to compile a production version of the OSP Toolkit with the Sun compiler (instead of gcc) without hardware acceleration, the compile command would be the following:

```
./osp_sdk_compile.ksh -p client -o use_spro -o no_hardware_accel
```

The compiler script will make the necessary adjustments and build the OSP Toolkit.

Custom Compilation in NT

Usually, changing only one or two parameters are 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:

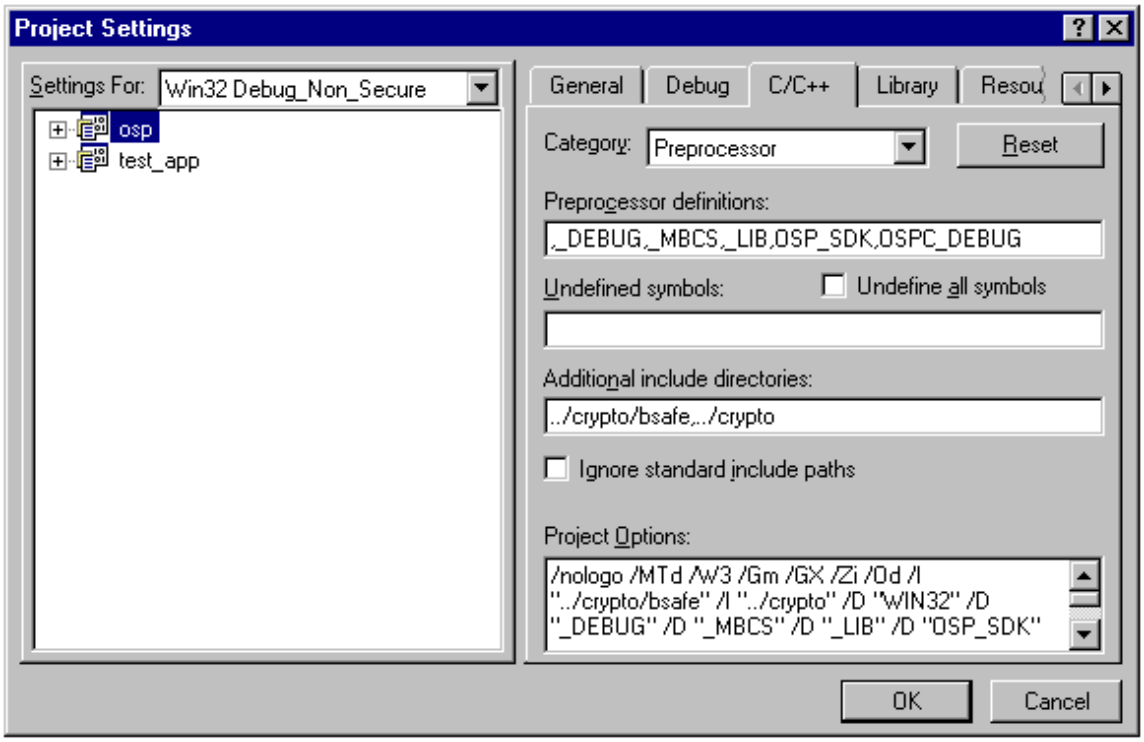
Menu: Project -> Settings ->
Tab: C/C++ -> category: Preprocessor

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, Debug_Non_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 NT, 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 indlude 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=betabe1/O=OSPServer
verify error:num=18:self signed certificate
verify return:1
depth=0 /CN=betabe1/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 is required to use the TransNexus OSP Test Server.

Can the OSP Toolkit use the BSAFE libraries from RSA Security? Can the OSP Toolkit use cryptographic hardware accelerators?

Yes. The OSP Toolkit can use BSAFE. Contact Transnexus at support@transnexus.com for details. As for cryptographic hardware accelerators, TransNexus has not thoroughly tested them with the OSP Toolkit. Check the System Development Kit (SDK) documents of the hardware accelerator you wish to use for more details about how to use them.

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 NT, 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
ssleay32.lib(s3_lib.obj) : error LNK2001: unresolved external symbol
__imp__qsort
libeay32.lib(stack.obj) : error LNK2001: unresolved external symbol
__imp__qsort
libeay32.lib(a_set.obj) : error LNK2001: unresolved external symbol
__imp__qsort
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
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__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(b_print.obj) : error LNK2001: unresolved external symbol
__imp__vsprintf
libeay32.lib(stack.obj) : error LNK2001: unresolved external symbol
__imp__bsearch
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(by_dir.obj) : error LNK2001: unresolved external symbol
__imp__stat
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/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 temp32out and temp32 directories, and start again at step 4 of the "Building the Toolkit in Windows NT" section again.

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=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.
MIIBTjunAmA0EQCm3c9pwWyjHIowF0/Ue92TMAbG9SqGSIb3DQEBBAUAMDsxJTAj
BgN7BAMTHG2zcHRlc3RzZXJ2ZXIudHJhbnYuZXh1cy5jb20xEjAQBgNVBAoTCU9T
UFNlcnZlcjAeFw0wMjAyMDQxOTQyMjdaFw0wMzAyMDUxOTQyMjdaMIGAMiswCQYD
VQQGEwJzUz5LMAwwA1UECBMCR0ExDDAKBgNVBAcTA0FsTDENMAsGA1UEfhMESklN
RDEPML0GA1UECxMGVGVzdGVyMQwwCgYDVkQDEwNKaW0xKDAmBgkqhkiG9w0BCQEW
GWppbS5kYWx0b25AdHJhbnNuZhh1cy5jb20wXDANBgkqhkiG9w0BAQEFAANLADBI
AkEAp3fkMb10pzruMuNaae4FOAq5UJlkpwbr13a+lFYH2H7s9ima/zQq6dC0ZGTi
6LJEhm/2iFKyXs3dYkgd7uXNAQIDAQABMA0GCSqGSIb3DQEBBAUAA0EAMcmeO+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 NT, 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..