Orbacus™

Using FreeSSL for Orbacus

Version 2.2, January 2007

Making Software Work Together™

Updated: February 1, 2007

Contents

Preface

The Orbacus Library

The Orbacus documentation library consists of the following books:

- Using Orbacus
- Using FreeSSL for Orbacus (this book)
- JThreads/C++
- Orbacus Notify
- .NET Connector Programmer's Guide

Using Orbacus

This manual describes how Orbacus implements the CORBA standard, and describes how to develop and maintain code that uses the Orbacus ORB. This is the primary developer's guide and reference for Orbacus.

Using FreeSSL for Orbacus

This manual describes the FreeSSL plug-in, which enables secure communications using the Orbacus ORB in both Java and C++.

JThreads/C++

This manual describes JThreads/C++, which is a high-level thread abstraction library that gives C++ programmers the look and feel of Java threads.

Orbacus Notify

This manual describes Orbacus Notify, an implementation of the Object Management Group's Notification Service specification.

.NET Connector Programmer's Guide

This manual describes the Orbacus .NET Connector, which enables transparent communication between clients running in a Microsoft .NET environment and servers running in a CORBA environment.

Audience

Manuals in the Orbacus library are written for intermediate to advanced level programmers who are:

- Experienced with Java or C++ programming
- Familiar with the CORBA standard and its specifications

These manuals do not teach the CORBA specification or CORBA programming in general, which are prerequisite skills. These manuals concentrate on how Orbacus implements the CORBA standard.

Getting the Latest Version

The latest updates to the Orbacus documentation can be found at http://www.iona.com/support/docs.

Compare the version dates on the web page for your product version with the date printed on the copyright page of the PDF edition of the book you are reading.

Searching the Orbacus Library

You can search the online documentation by using the **Search** box at the top right of the documentation home page:

http://www.iona.com/support/docs

To search a particular library version, browse to the required index page, and use the **Search** box at the top right.

You can also search within a particular book. To search within a HTML version of a book, use the **Search** box at the top left of the page. To search within a PDF version of a book, in Adobe Acrobat, select **Edit**|**Find**, and enter your search text.

Additional Resources

The IONA Knowledge Base (http://www.iona.com/support/knowledge_base/index.xml) contains helpful articles written by IONA experts about Orbacus and other products.

The IONA Update Center (http://www.iona.com/support/updates/index.xml) contains the latest releases and patches for IONA products.

If you need help with this or any other IONA product, go to IONA Online Support (http://www.iona.com/support/index.xml).

Comments, corrections, and suggestions on IONA documentation can be sent to docs-support@iona.com .

Document Conventions

Typographical conventions

This book uses the following typographical conventions:

`Fixed width`	Fixed width (Courier font) in normal text represents portions of code and literal names of items such as classes, functions, variables, and data structures. For example, text might refer to the `IT_Bus::AnyType` class. Constant width paragraphs represent code examples or information a system displays on the screen. For example: `#include <stdio.h>`
`Fixed width italic`	Fixed width italic words or characters in code and commands represent variable values you must supply, such as arguments to commands or path names for your particular system. For example: `% cd /users/`*`YourUserName`*
Italic	Italic words in normal text represent *emphasis* and introduce *new terms*.
Bold	Bold words in normal text represent graphical user interface components such as menu commands and dialog boxes. For example: the **User Preferences** dialog.

Keying Conventions

This book uses the following keying conventions:

No prompt	When a command's format is the same for multiple platforms, the command prompt is not shown.
`%`	A percent sign represents the UNIX command shell prompt for a command that does not require root privileges.
`#`	A number sign represents the UNIX command shell prompt for a command that requires root privileges.
`>`	The notation > represents the MS-DOS or Windows command prompt.
`...` . . .	Horizontal or vertical ellipses in format and syntax descriptions indicate that material has been eliminated to simplify a discussion.
`[]`	Brackets enclose optional items in format and syntax descriptions.
`{}`	Braces enclose a list from which you must choose an item in format and syntax descriptions.
`|`	In format and syntax descriptions, a vertical bar separates items in a list of choices enclosed in `{}` (braces). In graphical user interface descriptions, a vertical bar separates menu commands (for example, select **File|Open**).

Using FreeSSL for Orbacus

This chapter describes the FreeSSL plug-in, which enables secure communications using the Orbacus ORB in both Java and C++.

In this chapter

This chapter contains the following sections:

What is SSL?

Overview

The Secure Sockets Layer (SSL) protocol, developed by Netscape Communications Corporation, provides communications privacy over a network. It is designed to prevent eavesdropping, tampering, and message forgery. The FreeSSL plug-in enables secure communications using the Orbacus ORB in both Java and C++. The plug-in supports SSLv3 as defined in [1].

How Does It Work?

SSL uses symmetric cryptography for data communication (for example, DES). In symmetric cryptography, both parties use the same key to encrypt and decrypt data. This is different than asymmetric cryptography, in which different keys are used for encryption and decryption. The advantage of using symmetric cryptography for securing message traffic is that it operates much faster than asymmetric cryptography, thereby minimizing the overhead incurred by the use of a secure communication protocol.

Asymmetric cryptography, also known as public key cryptography (for example, RSA or DSS), is still used in the SSL protocol for authentication and key exchange. Using public key cryptography, each party has an associated public and private key. Data encrypted with the public key can only be decrypted with the private key, and vice versa. This allows a party to prove its identity by encrypting the data with its private key. As no other party has access to the private key, the data must have been sent by the true party.

Each peer is authenticated using an X.509 certificate [4]. Generally, a certificate will contain the user's name and public key and is signed by a trustworthy entity, the so-called Certificate Authority (CA).

Usually a chain of X.509 certificates are presented. The certificate at the head of the chain is the peer's certificate. Each certificate is signed by the next certificate in the chain. The certificate at the end of the chain is self-signed, and is generally the certificate of the Certificate Authority itself.

A certificate has an associated private key and passphrase. Without the private key is it not possible to use a certificate to prove identity. The passphrase protects the private key and is used to decrypt the private key at runtime.

Given a certificate, there must be some logic to determine whether this certificate is trusted. This is typically done against some certificate authority. A certificate authority is an organization that is responsible for issuing certificates to individuals. The choice of trusted certificate authorities is something that is best left up to the application. For instance, a company may issue certificates to all of their employees and only trust one certificate authority certificate.

The generation and signing of certificates is beyond the scope of this document. For the C++ plug-in please see [5], for the Java plug-in using iSaSiLk see [6].

The SSL protocol ensures that the connection between communicating parties is reliable. The integrity of the message data is verified using a keyed Message Authentication Code (MAC). The sender of a message uses a secure, one-way hash function (for example, SHA, MD5) to compute a unique MAC for the message. The receiver uses the same function to compute its own MAC, and then compares what it computed against the MAC computed by the sender. This means that corrupted or deliberately changed messages can be detected because the two MACs will not match.

Cipher Suites

A cipher suite [1] defines: The public key algorithm used for peer authentication and key exchange. The symmetric algorithm used for data encryption. The secure hash function for MAC computation. During the initial handshake, the client offers its set of supported cipher suites in its preferred order. The server responds by selecting one of the suites, or raising a handshake failure if they have none in common.

The following table summarizes the algorithms used by each cipher suite for key exchange, symmetric cryptography, and MAC calculation. Note that the SSL plug-in only supports the RSA and ADH suites.

Table 1: *Supported Cipher Suites*

Name	Key Alg	Symmetric Alg	MAC Calc
FSSL_RSA_EXPORT_WITH_NULL_MD5	RSA	None	MD5
FSSL_RSA_EXPORT_WITH_NULL_SHA	RSA	None	SHA
FSSL_RSA_EXPORT_WITH_RC4_40_MD5	RSA	RC4 (40 bits)	MD5
FSSL_RSA_WITH_RC4_128_MD5	RSA	RC4 (128 bits)	MD5

Table 1: *Supported Cipher Suites*

Name	Key Alg	Symmetric Alg	MAC Calc
FSSL_RSA_WITH_RC4_128_SHA	RSA	RC4 (128 bits)	SHA
FSSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	RSA	RC2 (40 bits)	MD5
FSSL_RSA_WITH_IDEA_CBC_SHA	RSA	IDEA (128 bits)	SHA
FSSL_RSA_EXPORT_WITH_DES40_CBC_SHA	RSA	DES (40 bits)	SHA
FSSL_RSA_WITH_DES_CBC_SHA	RSA	DES (56 bits)	SHA
FSSL_RSA_WITH_3DES_EDE_CBC_SHA	RSA	DES (168 bits)	SHA
FSSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA	RSA	DES (40 bits)	SHA
FSSL_DHE_RSA_WITH_DES_CBC_SHA	RSA	DES (56 bits)	SHA
FSSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA	RSA	DES (168 bits)	SHA
FSSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	DSS	DES (40 bits)	SHA
FSSL_DHE_DSS_WITH_DES_CBC_SHA	DSS	DES (56 bits)	SHA
FSSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DSS	DES (168 bits)	SHA
FSSL_DH_anon_EXPORT_WITH_RC4_40_MD5	ADH	RC4 (40 bits)	MD5
FSSL_DH_anon_WITH_RC4_128_MD5	ADH	RC4 (128 bits)	MD5
FSSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA	ADH	DES (40 bits)	SHA
FSSL_DH_anon_WITH_DES_CBC_SHA	ADH	DES (56 bits)	SHA
FSSL_DH_anon_WITH_3DES_EDE_CBC_SHA	ADH	DES (168 bits)	SHA

Note: Not all algorithms are supported in JSSE

Installation

Plug-in Installation

The FSSL plug-in is an implementation of the Orbacus Open Communications Interface (OCI) and is installed at runtime through configuration. For more general information on Orbacus configuration and the OCI please see the *Orbacus User Guide*.

Client Installation

The client side FSSL plug-in is installed as follows:

```
ooc.oci.client=fssliop [--seed FILE] [--backend IMPL]
    [--trace N]
```

The following options are supported:

`--seed FILE`	FreeSSL for C++ only. If specified, FreeSSL will use the contents of the file filename as random data to seed the OpenSSL (PRNG) Psuedo Random Number Generator. This may be necessary if the operating system doesn't have its own random data generator. (usually `/dev/random`) If no random data generator is found, and this property is not specified, FreeSSL will use a generic seeding algorithm.
`--backend IMPL`	FreeSSL for Java only. The Java version supports multiple third-party SSL toolkits which are identified to the plug-in during installation. Support for different third party SSL toolkits is provided through multiple back-end libraries where each library includes an implementation of the FSSLImpl interface. The `--backend` option accepts the name of the class implementing the FSSLImpl interface. By default the IAIK toolkit is used. Please see Appendix B for information on the supported SSL toolkits and the related back-end library. In this manual we will assume that the IAIK toolkit is being used.

`--trace N`	Sets the level of diagnostic output generated by the plug-in itself, and vendor-specific information from the underlying SSL toolkit. The default value is 0.

Server Installation

The server side FSSL plug-in is installed as shown below:

```
ooc.oci.server=fssliop
```

Note that FSSL servers must also install the client side plug-in.

Endpoint Configuration

Options

The configuration options for an FSSL endpoint are shown below:

```
fssliop [--backlog N] [--bind ADDR] [--host ADDR[,ADDR,...]]
    [--numeric] [--port N]
```

The following options are supported:

`--backlog N`	Specifies the length of the queue for incoming connection requests. Note that the operating system may override this setting if the value exceeds the maximum allowed.
`--bind ADDR`	Specifies the hostname or dotted decimal address of the network interface on which to bind the socket. If not specified, the POA Manager will bind its socket to all available network interfaces. This property is useful in situations where a host has several network interfaces, but the POA Manager should only listen for connections on a particular interface.
`--host ADDR[,ADDR,...]`	Specifies a list of one or more hostnames and/or dotted decimal addresses representing the addresses that should be advertised in IORs.
`--numeric`	If set, and if --host is not specified, then the canonical dotted decimal address is advertised in IORs. The default behavior is to use the canonical hostname, if possible.
`--port N`	Specifies the port number on which to bind the socket. If no port is specified the operating system selects an unused port automatically.

Command-Line Options

The FreeSSL plug-in defines the following command line options for both the C++ and the Java version of the plug-in:

`-FSSLbacklog N`	Equivalent to the --backlog endpoint option.
`-FSSLbind ADDR`	Equivalent to the --bind endpoint option.
`-FSSLhost ADDR[,ADDR,...]`	Equivalent to the --host endpoint option.
`-FSSLnumeric`	Equivalent to the --numeric endpoint option.
`-FSSLport N`	Equivalent to the --port endpoint option.

Static Linking

When statically linking a C++ application an explicit reference must be made to the FSSL plug-in in order to include the plug-in's modules. Shown below is the technique used by the sample programs in the fssl/demo subdirectory. Note that the code below is enclosed in guard macros that are only activated when statically linking. These macros are appropriate for both Unix and Windows. First, extra include files are necessary:

```
#if !defined(HAVE_SHARED) && !defined(FSSL_DLL)
#include <OB/OCI_init.h>
#include <FSSL/OCI_FSSLIOP_init.h>
#endif
Next, the plug-in must be registered prior to calling ORB_init():
#if !defined(HAVE_SHARED) && !defined(FSSL_DLL)
//
// When linking statically, we need to explicitly register the
// plug-in prior to ORB initialization
//
OCI::register_plugin("fssliop", OCI_init_fssliop);
#endif
```

URL Support

The FSSL plug-in supports corbaloc URLs with the following protocol syntax:

```
corbaloc:fssliop:host:port/object-key
```

The components of the URL are as follows:

`fssliop`	This selects the FSSL plug-in.
`host`	The hostname or IP address of the server.
`port`	The port on which the server is listening.
`object-key`	A stringified object key.

Contexts

What is a Context?

A context comprises three pieces of information: identity, trust decision, and a set of cipher suites. This information is necessary to establish an SSL connection from a client to a server and to allow a server to accept new SSL connections from clients. For anonymous communications only the set of cipher suites is necessary.

Context Creation

Contexts are managed via a context manager. A reference to the context manager is obtained by resolving the FSSLContextManager initial reference. To create a new context `FSSL::Manager::create_context` is called. This returns the ID of the newly created context.

```
// C++
FSSL::ContextID id = fsslManager -> create_context(
myChain, myKey, myPassPhrase, myDecider, myCiphers);
```

```
// Java
int id = fsslManager.create_context(
myChain, myKey, myPassPhrase, myDecider, myCiphers);
```

Contexts can also be created using a PKCS12 certificate file which contains a certificate chain and private key(s). To create a new context from a PKCS12 file, `FSSL::Manager::create_pkcs12_context` is called.

```
// C++
FSSL::ContextID id = fsslManager -> create_pkcs12_context(
        pkcs12_certificate, myPassPhrase, myDecider, myCiphers);
```

```
// Java
int id = fsslManager.create_pkcs12_context(pkcs12_certificate,
             myPassPhrase, myDecider, myCiphers);
```

To destroy a context call `FSSL::Manager::destroy_context`. Applications should be careful not to destroy contexts that are currently in use.

```
// C++
fsslManager -> destroy_context(id);
```

```
// Java
fsslManager.destroy_context(id);
```

Certificates

New X.509 certificates are created using the operation `FSSL::Manager::create_certificate`. An octet sequence containing a DER-encoded certificate should be passed as an argument.

```
// C++
FSSL::Certificate_var myCertificate =
fsslManager -> create_certificate(data);
```

```
// Java++
com.ooc.FSSL.Certificate myCertificate =
 fsslManager.create_certificate(data);
```

Since reading certificate data from a file is a typical use-case a helper method `FSSL::load_file` is provided. This takes a file name as the argument and returns an octet sequence.

```
// C++
FSSL::OctetSeq_var data = FSSL::load_file("mycert.der");
```

```
// Java
byte[] data = com.ooc.FSSL.FSSL.load_file("mycert.der");
```

Handling certificate data from a PKCS12 certificate file differs from DER certificate files. Data from the PKCS12 files is loaded directly into an octet sequence using FSSL::load_file and passed as a parameter to `FSSL::Manager::create_pkcs12_context`.

```
// C++
FSSL::OctetSeq_var pkcs12_data = FSSL::load_file("cert.p12");
FSSL::ContextID id = fsslManager -> create_pkcs12_context(
     pkcs12_data, myPassPhrase, myDecider, myCiphers);
```

```
// Java
byte[] pkcs12_data = com.ooc.FSSL.FSSL.load_file("cert.p12");
int id = fsslManager.create_pkcs12_context(pkcs12_data,
           myPassPhrase, myDecider, myCiphers);
```

Passphrase

The passphrase is an octet sequence. Again a typical use-case is that the passphrase is a string, therefore a helper method `FSSL_string_to_PassPhrase` is provided.

```
// C++
FSSL::PassPhrase_var myPassphrase =
   FSSL::string_to_PassPhrase("foobar");
```

```
// Java
byte[] myPassphrase =
   com.ooc.FSSL.FSSL.string_to_PassPhrase("foobar");
```

Cipher Suites

The context creation method is passed a sequence of cipher suite identifiers. A common use-case is to allow all non-anonymous ciphers. Therefore a helper method `FSSL::get_non_export_ciphers()` is provided.

```
// C++
FSSL::CipherSeq_var ciphers = FSSL::get_non_export_ciphers();
```

```
// Java
int[] ciphers = com.ooc.FSSL.FSSL.get_non_export_ciphers();
```

Three other helper methods are also provided. `FSSL::get_export_ciphers()` returns a sequence of all export RSA cipher suites (ciphers using keys that are less than 56 bits), `FSSL::get_RSA_ciphers()` returns a sequence of all RSA RSA cipher suites, `FSSL::get_DSS_ciphers()` returns a sequence of all DSS DSS cipher suites, and FSSL_get_ADH_ciphers returns a sequence of all ADH cipher suites.

If none of these helper methods supplies the desired functionality it is possible to manually construct a sequence of the cipher suites as follows:

```
// C++
FSSL::CipherSeq ciphers(2);
ciphers.length(2);
ciphers[0] = FSSL::RSA_WITH_RC4_128_MD5;
ciphers[1] = FSSL::RSA_WITH_RC4_128_SHA;
```

```
// Java
com.ooc.FSSL.Cipher[] ciphers =
{
    com.ooc.FSSL.Cipher.RSA_WITH_RC4_128_MD5.value,
    com.ooc.FSSL.Cipher.RSA_WITH_RC4_128_SHA.value,
};
```

Trust Decision

The application itself must be responsible for a determination of whether a certificate chain is trusted or not. To do this the application should provide an implementation of the TrustDecider interface.

```
interface TrustDecider
{
boolean is_trusted(in CertificateSeq chain);
};
```

The is_trusted method is called when each new connection is established or accepted. The trust decider can assume that the provided certificate chain is valid and good. That means that each certificate in the chain is signed by the next certificate and the last is self signed. If true is returned then the chain is trusted, and the connection may continue. If false is returned then the connection is rejected.

This example trust decider only trusts those certificates directly signed by some mythical certificate authority CA-X.

```
// C++
class MyTrustDecider : public FSSL::TrustDecider
{
//
// CA-X certificate
//
FSSL::Certificate_var cert_;

public:
MyTrustDecider(FSSL::Manager_ptr fsslManager)
{
    FSSL::OctetSeq_var data = FSSL::load_file("cax.der");
    cert_ = fsslManager -> create_certificate(data);
}

virtual CORBA::Boolean
is_trusted(const FSSL::CertificateSeq& chain)
{
    if(chain.length() == 2)
    return chain[1] -> is_signed_by(cert_);
    return false;
}
};
```

```
// Java
final class MyTrustDecider extends com.ooc.CORBA.LocalObject
implements com.ooc.FSSL.TrustDecider
{
//
// CA-X certificate
//
com.ooc.FSSL.Certificate cert_;

MyTrustDecider(com.ooc.FSSL.Manager fsslManager)
{
    cert_ = fsslManager.create_creatificate(
    com.ooc.FSSL.FSSL.load_file("cax.der"));
}

public bool
is_trusted(com.ooc.FSSL.Certificate[] chain)
{
    if(chain.length == 2)
    return chain[i].is_signed_by(cert_);
    return false;
}
}
```

Extending the Hello World Application

In order to demonstrate how to use the FreeSSL plug-in, the standard Hello World application included with Orbacus in the subdirectory demo/hello will be modified. The complete source code for this example is included with the FreeSSL distribution in the directory fssl/demo/hello.

In this chapter

This chapter contains the following sections:

Server Side Usage

Setting Identity

A server application must provide its identity using a context.

```
// C++
//
// Load the certificate chain
//
FSSL::CertificateSeq myCerts(2);
myCerts.length(2);
myCerts[0] = fsslManager -> create_certificate(
    FSSL::OctetSeq_var(FSSL::load_file("server.der")));

myCerts[1] = fsslManager -> create_certificate(
    FSSL::OctetSeq_var(FSSL::load_file("CAcert.der")));

//
// Create a new context with this certificate chain
//
FSSL::ContextID id = fsslManager -> create_context(
    myCerts,
    FSSL::OctetSeq_var(FSSL::loadFile("serverkey.der")),
    FSSL::PassPhrase_var(FSSL::string_to_PassPhrase("foobar")),
    myTrustDecider,
    FSSL::CipherSeq_var(FSSL::get_RSA_ciphers()));
```

```
// Java

//
// Load the certificate chain
//
com.ooc.FSSL.Certificate[] myCerts =
    new com.ooc.FSSL.Certificate[2];

myCerts[0] = fsslManager.create_certificate(
    com.ooc.FSSL.FSSL.load_file("server.der"));
myCerts[1] = fsslManager.create_certificate(
    com.ooc.FSSL.FSSL.load_file("ca.der"));

//
// Create the server context
//
int id = fsslManager.create_context(
    myCerts,
    com.ooc.FSSL.FSSL.load_file("serverkey.der"),
    com.ooc.FSSL.FSSL.string_to_PassPhrase("foobar"),
    myTrustDecider,
    com.ooc.FSSL.FSSL.get_RSA_ciphers());
```

This example defines the certificate chain for the server. The server's X.509 certificate will be obtained from the file server.der. This certificate is authenticated by the certificate in the file CAcert.der. The private key of the server's certificate is contained in the file serverkey.der and is decrypted using the passphrase foobar. In a real application it wouldn't be prudent to store the certificate's passphrase in plain text. Typically the pass-phrase should be requested from the user.

Once a context has been created, the next step is to call FSSL::create_poa_manager to initialize the server side of the FreeSSL connection. You can configure the RootPOA's POAManager simply by creating a POAManager name 'RootPOAManager'. Keep in mind that this step must be done prior to resolving the 'RootPOA' initial reference, otherwise the RootPOAManager will have already been created with the default configuration. The third and fourth arguments to FSSL::create_poa_manager are the reference to the FSSL::Manager and a

ContextID which should be associated with the POAManager to be created. The associated ContextID identifies the SSL identity the server will use when establishing connections.

```
// C++
PortableServer::POAManager_var poaManager =
FSSL::create_poa_manager(
"RootPOAManager", orb, fsslManager, id, props);
```

```
// Java
org.omg.PortableServer.POAManager poaManager =
com.ooc.FSSL.FSSL.create_poa_manager(
"RootPOAManager", orb, fsslManager, id, props);
```

Determining Peer Identity

The FSSL::Current interface can be used if the server needs to determine the identity of the peer that invoked the current operation.

First a reference to the FSSL::Current object must be retrieved.

```
// C++
FSSL::Current_var fsslCurrent =
FSSL::Current::_narrow(CORBA::Object_var(
    orb -> resolve_initial_references("FSSLCurrent")));
```

```
// Java
com.ooc.FSSL.Current fsslCurrent =
com.ooc.FSSL.CurrentHelper.narrow(
    orb.resolve_initial_references("FSSLCurrent"));
```

Now the FSSL::Current:get_peer_certificate_chain can be used to determine the identity of the caller:

```
// C++
FSSL::CertificateSeq_var chain =
    fsslCurrent -> get_peer_certificate_chain();
```

```
// Java
com.ooc.FSSL.X509Certificate[] chain =
    fsslCurrent.getPeerCertificateChain();
```

The negotiated cipher can also be determined using the `FSSL::Current` object.

```
// C++
FSSL::Cipher cipher = fsslCurrent -> get_peer_cipher();
```

```
// Java
com.ooc.FSSL.Cipher cipher = fsslCurrent.get_peer_cipher();
```

If this method is called outside of the context of a server method invocation a `FSSL::Current::NoContext` exception is raised. If the current connection is not an SSL connection then a `FSSL::Current::NoPeer` exception is raised.

Client Side Usage

Setting Identity

First a context must be created, as in the server case. Next a context policy must be created with the context id. Policies are a standard CORBA mechanism for controlling operational behaviour, and are considered to be immutable objects. That is, once they have been created, they may not be changed. The set of policies associated with an object reference are also considered to be immutable.

```
// C++
CORBA::Policy_var contextPolicy = fsslManager ->
    create_context_policy(id);
```

```
// Java
org.omg.CORBA.Policy contextPolicy =
    fsslManager.create_context_policy(id);
```

The CORBA standard provides three methods to associate policies with object references.

ORB Level Policies

The ORB level policies are managed using the ORB Policy Manager, which is resolved through the initial reference ORBPolicyManager.

```
// C++
CORBA::PolicyManager_var policyManager =
    CORBA::PolicyManager::_narrow(CORBA::Object_var(
    orb -> resolve_initial_references("ORBPolicyManager")));
```

```
// Java
org.omg.CORBA.PolicyManager policyManager =
    org.omg.CORBA.PolicyManagerHelper.narrow(
    orb.resolve_initial_references("ORBPolicyManager"));
```

Through this interface the current set of ORB level policies can be examined and changed. The set of ORB level policies will be associated with every new object reference that is created by that ORB.

Therefore, to associate a context policy with every object reference created by the ORB, the policy should be set on the ORB Policy Manager.

```
// C++
CORBA::PolicyList pl(1);
pl.length(1);
pl[0] = contextPolicy;
policyManger -> add_policy_overrides(pl);
```

```
// Java
org.omg.CORBA.Policy[] pl = new org.omg.CORBA.Policy[1];
pl[0] = contextPolicy;
policyManager.add_policy_overrides(pl);
```

Object Level Policies

Once object references have been created it is possible to create, a new object reference with a different set of associated policies by calling set_policy_overrides on the object reference. (In Java, set_policy_overrides is not actually called on the object, but on a delegate created from the object.)

```
// C++
CORBA::PolicyList pl(1);
pl.length(1);
pl[0] = contextPolicy;
CORBA::Object_var obj =
    myObj -> _set_policy_overrides(pl, CORBA::ADD_OVERRIDE);
```

```
// Java
org.omg.CORBA.Policy[] pl = new org.omg.CORBA.Policy[1];
pl[0] = contextPolicy;
com.ooc.CORBA.Delegate delegate = (com.ooc.CORBA.Delegate)
    ((org.omg.CORBA.portable.ObjectImpl)myObj)._get_delegate();
org.omg.CORBA.Object obj = delegate.set_policy_overrides(
pl, org.omg.CORBA.SetOverrideType.ADD_OVERRIDE);
```

Once set_policy_overrides has been called, the returned object reference will have a new set of associated policies. Note that the original object reference is not affected.

Thread Level Policies

A thread of execution in the application may have an associated set of policies. For the purposes of the SSL plug-in the context policy is not considered to be a thread level policy.

Full Example

The following is the full example:

```
// C++
FSSL::CertificateSeq myCerts(2);
myCerts.length(2);
myCerts[0] = fsslManager -> create_certificate(
    FSSL::OctetSeq_var(FSSL::loadFile("client.der")));
myCerts[1] = fsslManager -> create_certificate(
    FSSL::OctetSeq_var(FSSL::loadFile("CAcert.der")));
FSSL::ContextID id = fsslManager -> create_context(
    myCerts,
    FSSL::OctetSeq_var(FSSL::loadFile("clientkey.der")),
    FSSL::PassPhrase_var(FSSL::string_to_PassPhrase("foobar")),
    myTrustDecider,
    FSSL::CipherSeq_var(FSSL::getDefaultCiphers()));
CORBA::PolicyManager_var policyManager =
    CORBA::PolicyManager::_narrow(CORBA::Object_var(
    orb -> resolve_initial_references("ORBPolicyManager")));
CORBA::PolicyList pl(1);
pl.length(1);
pl[0] = fsslManager -> create_context_policy(id);
policymanger -> add_policy_overrides(pl);
```

```
// Java
com.ooc.FSSL.Certificate[] myCerts = new
   com.ooc.FSSL.Certificate[2];
myCerts[0] = fsslManager.create_certificate(
    com.ooc.FSSL.FSSL.load_file("client.der"));
myCerts[1] = fsslManager.create_certificate(
    com.ooc.FSSL.FSSL.load_file("ca.der"));
int id = fsslManager.create_context(
    myCerts,
    com.ooc.FSSL.FSSL.load_file("clientkey.der"),
    com.ooc.FSSL.FSSL.string_to_PassPhrase("foobar"),
    myTrustDecider,
    com.ooc.FSSL.FSSL.get_default_ciphers());
org.omg.CORBA.PolicyManager policyManager =
    org.omg.CORBA.PolicyManagerHelper.narrow(
    orb.resolve_initial_references("ORBPolicyManager"));
org.omg.CORBA.Policy[] pl = new org.omg.CORBA.Policy[1];
pl[0] = fsslManager.create_context_policy(id);
policyManager.add_policy_overrides(pl);
```

Determining Peer Identity

Before the client can determine the identity of the peer it must first get the OCI::FSSLIOP::TransportInfo. The client accomplishes this by calling _non_existent() on the object reference to force the connection and then narrowing the OCI::TransportInfo.

```
// C++
OCI::FSSLIOP::TransportInfo_var fssliopInfo;
if(!obj -> _non_existent())
{
OCI::TransportInfo_var info obj -> _get_oci_transport_info();
fssliopInfo = OCI::FSSLIOP::TransportInfo::_narrow(info);
}
```

```
// Java
com.ooc.OCI.FSSLIOP.TransportInfo fssliopInfo = null;
if(!obj._non_existent())
{
org.omg.CORBA.portable.ObjectImpl objImpl =
(org.omg.CORBA.portable.ObjectImpl)obj;
com.ooc.CORBA.Delegate objDelegate =
(com.ooc.CORBA.Delegate)objImpl._get_delegate();

com.ooc.OCI.TransportInfo info =
objDelegate.get_oci_transport_info();
fssliopInfo =
   com.ooc.OCI.FSSLIOP.TransportInfoHelper.narrow(info);
}
```

Once a reference to the FSSLIOP transport information is aquired, OCI::FSSLIOP::TransportInfo::certificate_chain can be used to determine the identity of the caller:

```
// C++
FSSL::CertificateSeq_var chain =
fssliopInfo -> certificate_chain();
```

```
// Java
com.ooc.FSSL.Certificate[] chain =
fssliopInfo.certificate_chain();
```

The negotiated cipher can be determined using the OCI::FSSLIOP::TransportInfo::negotiated_cipher.

```
// C++
FSSL::Cipher cipher = fssliopInfo -> negotiated_cipher();
```

```
// Java
com.ooc.FSSL.Cipher cipher = fssliopInfo.negotiated_cipher();
```

Preventing Connections to Secure/Insecure Servers

In developing your applications you may want to restrict the servers to which your proxy will connect. For instance, you may want to connect only with secure servers, or alternatively only with insecure servers.

To do this, a ProtocolPolicy policy must be used. The ProtocolPolicy is used to restrict the protocol that will be used to establish communications. By default, after initializing the FreeSSL plug-in, a protocol policy with a value of OCI::FSSLIOP::PLUGIN_ID is set as an ORB level policy. Therefore, only secure connections will be established unless this is overridden. To allow an object reference to use IIOP the protocol policy can be overridden on the reference as follows:

```
// C++
CORBA::Any any;
any <<= OCI::IIOP::PLUGIN_ID;
CORBA::PolicyList pl(1);
pl.length(1);
pl[0] = orb -> create_policy(OB::PROTOCOL_POLICY_ID, any);
CORBA::Object_var myObj = obj -> _set_policy_overrides(
    pl, CORBA::ADD_OVERRIDE);
```

```
// Java
org.omg.CORBA.Any any = orb_.create_any();
   any.insert_ulong(com.ooc.OCI.IIOP.PLUGIN_ID.value);
org.omg.CORBA.Policy[] pl = new org.omg.CORBA.Policy[1];
pl[0] = orb.create_policy(
    com.ooc.OB.PROTOCOL_POLICY_ID.value, any);
com.ooc.CORBA.Delegate delegate = (com.ooc.CORBA.Delegate)
    ((org.omg.CORBA.portable.ObjectImpl)myObj)._get_delegate();
org.omg.CORBA.Object obj = delegate.set_policy_overrides(
    myObj, pl, org.omg.CORBA.SetOverrideType.ADD_OVERRIDE);
```

If it is necessary to revert to a secure transport again for establishing further connections (for instance: case of a client creating successive connections to secure and insecure servers), simply reapply the OCI::FSSLIOP::PLUGIN_ID protocol policy as needed.

Complete Example

Certificates

First the certificates must be created for both the client and the server. For a real world application the certificates will most likely be provided by an actual certificate authority. However, for the purposes of this demo we'll generate the certificates by hand.

OpenSSL

First create a certificate authority.

```
> cd /tmp
> CA.sh -newca
```

Next create a certificate request and sign the request using the new certificate authority. Use passphrase blahblah.

```
> CA.sh -newreq
> CA.sh -sign
```

Next the private key must be converted from PEM format to PKCS#8 DER format.

```
> openssl pkcs8 -outform DER -in newreq.pem -out newkey.der -topk8
```

Finally, the new certificate and the CA's certificate must be converted from PEM to DER encoding.

```
> openssl x509 -outform DER -in newcert.pem -out newcert.der
> openssl x509 -outform DER -in demoCA/cacert.pem -out cacert.der
```

This must be done to create two sets of certificates and private keys, one set for the server and one set for the client. Store the client set in client.der, and client.key. Store the server set in server.der and server.key. The CA's certificate should be in ca.der.

When creating certificates it's necessary to provide identity information. For the Server, use Server for the common name section of the certificate's Subject field. This will be used later for trust decisions.

iSaSiLk

For this toolkit an application must be written to generate the certificates. Since this is beyond the scope of the manual the reader is advised to consult the application fssl/demo/hello/GenCerts.java bundled with the FreeSSL for Java distribution.

Client Side

main

First initialize the ORB.

```
// C++

int
main(int argc, char* argv[], char*[])
{
int status = EXIT_SUCCESS;
CORBA::ORB_var orb;

    try
    {
    orb = CORBA::ORB_init(argc, argv);
status = run(orb, argc, argv);
}
catch(const CORBA::Exception& ex)
{
cerr << ex << endl;
status = EXIT_FAILURE;
}

if(!CORBA::is_nil(orb))
{
try
{
orb -> destroy();
}
catch(const CORBA::Exception& ex)
{
cerr << ex << endl;
status = EXIT_FAILURE;
}
}

return status;
}
```

```
// Java

public static void
main(String args[])
{
int status = 0;
org.omg.CORBA.ORB orb = null;

java.util.Properties props = System.getProperties();
    props.put("org.omg.CORBA.ORBClass", "com.ooc.CORBA.ORB");
    props.put("org.omg.CORBA.ORBSingletonClass",
      "com.ooc.CORBA.ORB");

    try
    {
    orb = org.omg.CORBA.ORB.init(args, props);
status = run(orb, args);
}
catch(Exception ex)
{
ex.printStackTrace();
status = 1;
}

if(orb != null)
{
try
{
((com.ooc.CORBA.ORB)orb).destroy();
}
catch(Exception ex)
{
ex.printStackTrace();
status = 1;
}
}

System.exit(status);
}
```

run

Next obtain a reference to the FSSL Context Manager.

```
// C++

int
run(CORBA::ORB_ptr orb, int argc, char* argv[])
{
    OBCORBA::ORB_var oborb = OBCORBA::ORB::_narrow(orb);

    //
    // Obtain the ORB's property set
    //
    OB::Properties_var props = oborb -> properties();

    //
    // Resolve the FSSL Context Manager
    //
    CORBA::Object_var fsslManagerObj =
    orb -> resolve_initial_references("FSSLContextManager");
    FSSL::Manager_var fsslManager =
   FSSL::Manager::_narrow(fsslManagerObj);
```

```
// Java

static int
run(org.omg.CORBA.ORB orb, String[] args)
throws org.omg.CORBA.UserException
{
//
    // Obtain the ORB's property set
    //
    java.util.Properties props =
   ((com.ooc.CORBA.ORB)orb).properties();

    //
    // Resolve the FSSL Context Manager
    //
    com.ooc.FSSL.Manager fsslManager =
    com.ooc.FSSL.ManagerHelper.narrow(
    orb.resolve_initial_references("FSSLContextManager"));
```

Next the client's certificate chain must be constructed.

```
// C++

    //
    // Create the clients certificate chain
    //
    FSSL::Certificate_var clientCert =
    fsslManager -> create_certificate(
        FSSL::OctetSeq_var(FSSL::load_file("client.der")));
    FSSL::Certificate_var caCert =
    fsslManager -> create_certificate(
    FSSL::OctetSeq_var(FSSL::load_file("ca.der")));

FSSL::CertificateSeq chain;
chain.length(2);
chain[0] = clientCert;
chain[1] = caCert;
```

```
// Java

    //
    // Create the client certificate chain
    //
    com.ooc.FSSL.Certificate clientCert =
    fsslManager.create_certificate(
        com.ooc.FSSL.FSSL.load_file("client.der"));
    com.ooc.FSSL.Certificate caCert =
    fsslManager.create_certificate(
        com.ooc.FSSL.FSSL.load_file("ca.der"));

    com.ooc.FSSL.Certificate[] chain =
    new com.ooc.FSSL.Certificate[2];
    chain[0] = clientCert;
    chain[1] = caCert;
```

Once that has been done a context must be created. For this demo all RSA ciphers can be used. The implementation of the TrustDecider will come a little later.

```
// C++

//
// Create the client context
//
FSSL::ContextID id = fsslManager -> create_context(
chain,
FSSL::OctetSeq_var(FSSL::load_file("client.key")),
FSSL::PassPhrase_var(FSSL::string_to_PassPhrase("blahblah")),
FSSL::TrustDecider_var(new TrustDecider_impl(caCert)),
FSSL::CipherSeq_var(FSSL::get_RSA_ciphers()));
```

```
// Java

    //
    // Create the client context
    //
    int id = fsslManager.create_context(
    chain,
    com.ooc.FSSL.FSSL.load_file("client.key"),
    com.ooc.FSSL.FSSL.string_to_PassPhrase("blahblah"),
    new ClientTrustDecider(caCert),
    com.ooc.FSSL.FSSL.get_RSA_ciphers());
```

After that the context should be set as the default context for all object references.

```
// C++

//
// Set this as the default context for all object references
//
fsslManager -> set_context(id);
```

```
// Java

    //
    // Set this as the default context for all object references
    //
    fsslManager.set_context(id);
```

After this has been done the remainder of run will be the same as the original demo.

```
// C++

//
// Get "hello" object
//
CORBA::Object_var obj = orb ->
   string_to_object("relfile:/Hello.ref");
if(CORBA::is_nil(obj))
{
cerr << argv[0] << ": cannot read IOR from Hello.ref" << endl;
    return EXIT_FAILURE;
}

Hello_var hello = Hello::_narrow(obj);
assert(!CORBA::is_nil(hello));

//
// Main loop
//
cout << "Enter 'h' for hello or 'x' for exit:\n";
char c;
do
{
    cout << "> ";
    cin >> c;
    if(c == 'h')
    hello -> say_hello();
}
while(cin.good() && c != 'x');

return EXIT_SUCCESS;
}
```

```
// Java

//
// Get "hello" object
//
CORBA::Object_var obj = orb ->
   string_to_object("relfile:/Hello.ref");
if(CORBA::is_nil(obj))
{
cerr << argv[0] << ": cannot read IOR from Hello.ref" << endl;
    return EXIT_FAILURE;
}

Hello_var hello = Hello::_narrow(obj);
assert(!CORBA::is_nil(hello));

//
// Main loop
//
cout << "Enter 'h' for hello or 'x' for exit:\n";
char c;
do
{
    cout << "> ";
    cin >> c;
    if(c == 'h')
    hello -> say_hello();
}
while(cin.good() && c != 'x');

return EXIT_SUCCESS;
}
```

The Trust Decider

The TrustDecider implementation for the demo will be extremely simple. It will trust only those certificates directly signed by the provided CA. To implement the TrustDecider the class FSSL_TrustDecider must be implemented. In addition on the client side only the server will be trusted.

```
// C++

class TrustDecider_impl : public FSSL::TrustDecider
```

```
// Java

class ClientTrustDecider extends com.ooc.CORBA.LocalObject
    implements com.ooc.FSSL.TrustDecider
```

Next the private members and constructor.

```
// C++

    FSSL::Certificate_var ca_;

public:

    TrustDecider_impl(FSSL::Certificate_var ca)
    : ca_(FSSL::Certificate::_duplicate(ca))
    {
    }
```

```
// Java

    private com.ooc.FSSL.Certificate ca_;

    ClientTrustDecider(com.ooc.FSSL.Certificate ca)
    {
    ca_ = ca;
    }
```

Next, is_trusted must be implemented.

```
// C++

virtual CORBA::Boolean
is_trusted(const FSSL::CertificateSeq& chain)
```

```
// Java

public boolean
is_trusted(com.ooc.FSSL.Certificate[] chain)
```

This method should ensure that the CA in the certificate chain is the CA provided by the constructor. To do that it should be verfied that the CA has signed the last certificate in the chain (since CA certificates are self signed), and that the subject distinguished names are the same. In addition the common name portion of the server side certificate will be examined to ensure that only the server is accepted. Note that for a real world example more than just the common name should be validated, since it's possible that the common name is the same for two certificates.

```
// C++

CORBA::String_var serverDN = chain[0] -> subject_DN();
if(strstr(serverDN, "CN=Server/") == 0)
    return false;
if(chain.length() == 2 && chain[1] -> is_signed_by(ca_))
{
    CORBA::String_var dn1 = chain[1] -> subject_DN();
    CORBA::String_var dn2 = ca_ -> subject_DN();
    if(strcmp(dn1, dn2) == 0)
    return true;
}
return false;
```

```
// Java

String serverDN = chain[0].subject_DN();
if(serverDN.indexOf("CN=Server,") == -1)
return false;

if(chain.length == 2 && chain[1].is_signed_by(ca_))
{
String dn1 = chain[1].subject_DN();
String dn2 = ca_.subject_DN();
if(dn1.equals(dn2))
return true;
}
return false;
```

Server Side

main

First initialize the ORB.

```
// C++

int
main(int argc, char* argv[], char*[])
{
int status = EXIT_SUCCESS;
CORBA::ORB_var orb;

    try
    {
    orb = CORBA::ORB_init(argc, argv);
status = run(orb, argc, argv);
}
catch(const CORBA::Exception& ex)
{
cerr << ex << endl;
status = EXIT_FAILURE;
}

if(!CORBA::is_nil(orb))
{
try
{
orb -> destroy();
}
catch(const CORBA::Exception& ex)
{
cerr << ex << endl;
status = EXIT_FAILURE;
}
}

return status;
}
```

```
// Java

public static void
main(String args[])
{
int status = 0;
org.omg.CORBA.ORB orb = null;

java.util.Properties props = System.getProperties();
    props.put("org.omg.CORBA.ORBClass", "com.ooc.CORBA.ORB");
    props.put("org.omg.CORBA.ORBSingletonClass",
      "com.ooc.CORBA.ORB");

    try
    {
    orb = org.omg.CORBA.ORB.init(args, props);
status = run(orb, args);
}
catch(Exception ex)
{
ex.printStackTrace();
status = 1;
}

if(orb != null)
{
try
{
((com.ooc.CORBA.ORB)orb).destroy();
}
catch(Exception ex)
{
ex.printStackTrace();
status = 1;
}
}

System.exit(status);
}
```

run

Next obtain a reference to the FSSL Context Manager.

```
// C++

int
run(CORBA::ORB_ptr orb, int argc, char* argv[])
{
    OBCORBA::ORB_var oborb = OBCORBA::ORB::_narrow(orb);

    //
    // Obtain the ORB's property set
    //
    OB::Properties_var props = oborb -> properties();

    //
    // Resolve the FSSL Context Manager
    //
    CORBA::Object_var fsslManagerObj =
    orb -> resolve_initial_references("FSSLContextManager");
    FSSL::Manager_var fsslManager =
   FSSL::Manager::_narrow(fsslManagerObj);
```

```
// Java

static int
run(org.omg.CORBA.ORB orb, String[] args)
throws org.omg.CORBA.UserException
{
//
    // Obtain the ORB's property set
    //
    java.util.Properties props =
   ((com.ooc.CORBA.ORB)orb).properties();

    //
    // Resolve the FSSL Context Manager
    //
    com.ooc.FSSL.Manager fsslManager =
    com.ooc.FSSL.ManagerHelper.narrow(
    orb.resolve_initial_references("FSSLContextManager"));
```

Next the certificate chain for the server must be created. This is exactly the same procedure as for the client.

```
// C++

    //
    // Create the servers certificate chain
    //
    FSSL::Certificate_var serverCert =
    fsslManager -> create_certificate(
        FSSL::OctetSeq_var(FSSL::load_file("server.der")));
    FSSL::Certificate_var caCert =
    fsslManager -> create_certificate(
    FSSL::OctetSeq_var(FSSL::load_file("ca.der")));

FSSL::CertificateSeq chain;
chain.length(2);
chain[0] = serverCert;
chain[1] = caCert;
```

```
// Java

    //
    // Create the server certificate chain
    //
    com.ooc.FSSL.Certificate serverCert =
    fsslManager.create_certificate(
        com.ooc.FSSL.FSSL.load_file("server.der"));
    com.ooc.FSSL.Certificate caCert =
    fsslManager.create_certificate(
        com.ooc.FSSL.FSSL.load_file("ca.der"));

    com.ooc.FSSL.Certificate[] chain =
    new com.ooc.FSSL.Certificate[2];
    chain[0] = serverCert;
    chain[1] = caCert;
```

Once that has been done a context must be created. For this demo all RSA ciphers can be used. The implementation of the TrustDecider will come a little later.

```
// C++

//
// Create the server context
//
FSSL::ContextID id = fsslManager -> create_context(
chain,
FSSL::OctetSeq_var(FSSL::load_file("server.key")),
FSSL::PassPhrase_var(FSSL::string_to_PassPhrase("blahblah")),
FSSL::TrustDecider_var(new TrustDecider_impl(caCert)),
FSSL::CipherSeq_var(FSSL::get_RSA_ciphers()));
```

```
// Java

    //
    // Create the server context
    //
    int id = fsslManager.create_context(
    chain,
    com.ooc.FSSL.FSSL.load_file("server.key"),
    com.ooc.FSSL.FSSL.string_to_PassPhrase("blahblah"),
    new ClientTrustDecider(caCert),
    com.ooc.FSSL.FSSL.get_RSA_ciphers());
```

Once the SSL context has been created, the POAManager can be initialized and the RootPOA resolved.

```
// C++

//
// Create the POA Manager
//
PortableServer::POAManager_var poaManager =
FSSL::create_poa_manager(
    "RootPOAManager", orb, fsslManager, id, props);

//
// Resolve Root POA
//
CORBA::Object_var poaObj =
orb -> resolve_initial_references("RootPOA");
PortableServer::POA_var rootPOA =
PortableServer::POA::_narrow(poaObj);
```

```
// Java

//
    // Create the POA Manager
    //
    org.omg.PortableServer.POAManager poaManager =
    com.ooc.FSSL.FSSL.create_poa_manager(
    "RootPOAManager", orb, fsslManager, id, props);

//
// Resolve Root POA
//
org.omg.PortableServer.POA root =
org.omg.PortableServer.POAHelper.narrow(
orb.resolve_initial_references("RootPOA"));
```

After this has been done the remainder of run will be the same as the original demo.

```
// C++

//
// Create implementation object
//
Hello_impl* helloImpl = new Hello_impl();
PortableServer::ServantBase_var servant = helloImpl;
Hello_var hello = helloImpl -> _this();

//
// Save reference
//
CORBA::String_var s = orb -> object_to_string(hello);

const char* refFile = "Hello.ref";
ofstream out(refFile);
if(out.fail())
{
    cerr << argv[0] << ": can't open `" << refFile << "': "
     << strerror(errno) << endl;
    return EXIT_FAILURE;
}

out << s << endl;
out.close();

//
// Run implementation
//
cout << "Server is ready." << endl;
poaManager -> activate();
orb -> run();

return EXIT_SUCCESS;
}
```

```
// Java

//
// Create implementation object
//
Hello_impl helloImpl = new Hello_impl();
    Hello hello = helloImpl._this(orb);

//
// Save reference
//
try
{
String ref = orb.object_to_string(hello);
String refFile = "Hello.ref";
java.io.FileOutputStream file =
new java.io.FileOutputStream(refFile);
java.io.PrintWriter out = new java.io.PrintWriter(file);
out.println(ref);
out.flush();
file.close();
}
catch(java.io.IOException ex)
{
System.err.println("hello.Server: can't write to `" +
   ex.getMessage() + "'");
return 1;
}

//
// Run implementation
//
    System.out.println("Server is ready.");
poaManager.activate();
orb.run();

return 0;
}
```

Trust Decider

The trust decider for the server is slightly different in that the distinguished name of the client is not validated since the server accepts connections from any client validated by the CA.

```
// C++

if(chain.length() == 2 && chain[1] -> is_signed_by(ca_))
{
    CORBA_String_var dn1 = chain[1] -> subject_DN();
    CORBA_String_var dn2 = ca_ -> subject_DN();
    if(strcmp(dn1, dn2) == 0)
    return true;
}
return false;
```

```
// Java

if(chain.length == 2 && chain[1].is_signed_by(ca_))
{
String dn1 = chain[1].subject_DN();
String dn2 = ca_.subject_DN();
if(dn1.equals(dn2))
return true;
}
return false;
```

FSSL Definitions

ADH
The anonymous Diffie-Hellman public-key algorithm, see [9].

ASN.1
Abstract Syntax Notation One, see [14].

DER
Distinguished Encoding Rules for ASN.1, see [4].

DES
Data Encryption Standard, see [12].

DSS
The Digital Signature Standard, see [11]

IDEA
International Data Encryption Algorithm, see [11].

MD5
RSA Data Security, Inc.'s MD5 message-digest algorithm, see [8].

PEM
Internet Privacy-Enhanced Mail, see [14]-[17].

PKCS#8
Private-Key Information Syntax Standard, see [18].

RC2, RC4
Rivest's Ciphers, variable-key-size encryption algorithms, see [11].

RSA
The RSA public-key cryptosystem, see [3].

SHA
Secure Hash Algorithm, see [7].

Toolkits Supported by FSSL

Supported Toolkits

Both FreeSSL for C++ and Java require third-party SSL toolkits to operate.

DISCLAIMER: IONA Technologies does not assume any responsibility for the purchase or licensing of any third-party product that is required to work with a particular version of the SSL plug-in. Any licensing issues that arise as a result of the use of any third party product is the sole responsibility of the purchaser.

OpenSSL

FreeSSL for C++ requires OpenSSL 0.9.7g. This is a public domain implementation of the Secure Sockets Layer version 3.0. Please see http://www.openssl.org for more information on this product.

IAIK iSaSiLk

FreeSSL for Java supports version 3.04 of the IAIK-iSaSiLk SSL toolkit and version 3.0 (or equivalent Applet Edition) of the IAIK JCE. This is an excellent SSL toolkit available from the IAIK-Java Group. Please see http://jce.iaik.tugraz.at/ for more information on this product.

JSSE

FreeSSL for Java supports the JSSE toolkit. JSSE is available from Sun and is bundled with JDK 1.4 and above. Please see `http://java.sun.com` for more information on this product.

FSSL Reference

This appendix documents the FSSL interfaces.

In this appendix

This appendix contains the following sections:

Module CORBA

Interface Index

Current

Provides information on the current connection.

Policy

Provides information on the current policy.

Aliases

PolicyList

```
typedef sequence<Policy> PolicyList;
```

PolicyType

```
typedef unsigned long PolicyType;
```

PolicyTypeSeq

```
typedef sequence<PolicyType> PolicyTypeSeq;
```

Module FSSL

Overview

The FSSL plug-in interfaces. This module allows for the configuration of the Secure Sockets Layer OCI plug-in.

Interface Index

Certificate

X509 Certificate Interface

ContextPolicy

Context Policy Interface

Current

Provides information on the current connection.

Manager

Manager Interface

TrustDecider

TrustDecider Interface allows users to provide custom certificate chain trust algorithms

Constants

BAD_CIPHER

```
const Cipher BAD_CIPHER = 0;
```

Identifies an invalid cipher

CONTEXT_POLICY

```
const CORBA::PolicyType CONTEXT_POLICY = 100;
```

Identifies the ContextPolicy.

DHE_DSS_EXPORT_WITH_DES40_CBC_SHA

```
const Cipher DHE_DSS_EXPORT_WITH_DES40_CBC_SHA = 14;
```

Key Exchange Algorithm DHE_DSS

Symmetric Encryption Algorithm DES(40)

MAC Encoding SHA

DHE_DSS_WITH_3DES_EDE_CBC_SHA

```
const Cipher DHE_DSS_WITH_3DES_EDE_CBC_SHA = 16;
```

Key Exchange Algorithm DHE_DSS

Symmetric Encryption Algorithm DES(168)

MAC Encoding SHA

DHE_DSS_WITH_DES_CBC_SHA

`const Cipher DHE_DSS_WITH_DES_CBC_SHA = 15;`

Key Exchange Algorithm DHE_DSS

Symmetric Encryption Algorithm DES(56)

MAC Encoding SHA

DHE_RSA_EXPORT_WITH_DES40_CBC_SHA

`const Cipher DHE_RSA_EXPORT_WITH_DES40_CBC_SHA = 11;`

Key Exchange Algorithm DHE_RSA

MAC Encoding SHA

DHE_RSA_WITH_3DES_EDE_CBC_SHA

`const Cipher DHE_RSA_WITH_3DES_EDE_CBC_SHA = 13;`

Key Exchange Algorithm DHE_RSA

Symmetric Encryption Algorithm DES(168)

MAC Encoding SHA

DHE_RSA_WITH_DES_CBC_SHA

`const Cipher DHE_RSA_WITH_DES_CBC_SHA = 12;`

Key Exchange Algorithm DHE_RSA

Symmetric Encryption Algorithm DES(56)

MAC Encoding SHA

DH_anon_EXPORT_WITH_DES40_CBC_SHA

`const Cipher DH_anon_EXPORT_WITH_DES40_CBC_SHA = 19;`

Key Exchange Algorithm DH

Symmetric Encryption Algorithm DES(40)

MAC Encoding SHA

DH_anon_EXPORT_WITH_RC4_40_MD5

`const Cipher DH_anon_EXPORT_WITH_RC4_40_MD5 = 17;`

Key Exchange Algorithm DH

Symmetric Encryption Algorithm RC4(40)

MAC Encoding MD5

DH_anon_WITH_3DES_EDE_CBC_SHA

`const Cipher DH_anon_WITH_3DES_EDE_CBC_SHA = 21;`

Key Exchange Algorithm DH

Symmetric Encryption Algorithm DES(168)

MAC Encoding SHA

DH_anon_WITH_DES_CBC_SHA

`const Cipher DH_anon_WITH_DES_CBC_SHA = 20;`

Key Exchange Algorithm DH

Symmetric Encryption Algorithm DES(56)

MAC Encoding SHA

DH_anon_WITH_RC4_128_MD5

`const Cipher DH_anon_WITH_RC4_128_MD5 = 18;`

Key Exchange Algorithm DH

Symmetric Encryption Algorithm RC4(128)

MAC Encoding MD5

RSA_EXPORT_WITH_DES40_CBC_SHA

`const Cipher RSA_EXPORT_WITH_DES40_CBC_SHA = 8;`

Key Exchange Algorithm RSA

Symmetric Encryption Algorithm DES(40)

MAC Encoding SHA

RSA_EXPORT_WITH_NULL_MD5

`const Cipher RSA_EXPORT_WITH_NULL_MD5 = 1;`

Key Exchange Algorithm RSA

Symmetric Encryption Algorithm NULL

MAC Encoding MD5

RSA_EXPORT_WITH_NULL_SHA

`const Cipher RSA_EXPORT_WITH_NULL_SHA = 2;`

Key Exchange Algorithm RSA

Symmetric Encryption Algorithm NULL

MAC Encoding MD5

RSA_EXPORT_WITH_RC2_CBC_40_MD5

`const Cipher RSA_EXPORT_WITH_RC2_CBC_40_MD5 = 6;`

Key Exchange Algorithm RSA

Symmetric Encryption Algorithm RC2(40)

MAC Encoding MD5

RSA_EXPORT_WITH_RC4_40_MD5

```
const Cipher RSA_EXPORT_WITH_RC4_40_MD5 = 3;
```

Key Exchange Algorithm RSA

Symmetric Encryption Algorithm RC4(40)

MAC Encoding MD5

RSA_WITH_3DES_EDE_CBC_SHA

```
const Cipher RSA_WITH_3DES_EDE_CBC_SHA = 10;
```

Key Exchange Algorithm RSA

Symmetric Encryption Algorithm DEC(168)

MAC Encoding SHA

RSA_WITH_DES_CBC_SHA

```
const Cipher RSA_WITH_DES_CBC_SHA = 9;
```

Key Exchange Algorithm RSA

Symmetric Encryption Algorithm DEC(56)

MAC Encoding SHA

RSA_WITH_IDEA_CBC_SHA

```
const Cipher RSA_WITH_IDEA_CBC_SHA = 7;
```

Key Exchange Algorithm RSA

Symmetric Encryption Algorithm IDEA(128)

MAC Encoding SHA

RSA_WITH_RC4_128_MD5

```
const Cipher RSA_WITH_RC4_128_MD5 = 4;
```

Key Exchange Algorithm RSA

Symmetric Encryption Algorithm RC4(128)

MAC Encoding MD5

RSA_WITH_RC4_128_SHA

```
const Cipher RSA_WITH_RC4_128_SHA = 5;
```

Key Exchange Algorithm RSA

Symmetric Encryption Algorithm RC4(128)

MAC Encoding SHA

Aliases

CertificateSeq

```
typedef sequence<Certificate> CertificateSeq;
```

Alias for an X509 Certificate Chain

Cipher

```
typedef unsigned long Cipher;
```

An alias for a cipher suite

CipherSeq

```
typedef sequence<Cipher> CipherSeq;
```

Alias for a sequence of Ciphers

ContextID

```
typedef unsigned long ContextID;
```

Alias for Context ID.

OctetSeq

```
typedef sequence<octet> OctetSeq;
```

Alias for sequences of octets

PassPhrase

```
typedef sequence<octet> PassPhrase;
```

Alias for a PassPhrase

PrivateKey

```
typedef sequence<octet> PrivateKey;
```

Alias for a PrivateKey

Module IOP

Constants

CodeSets

```
const ServiceId CodeSets = 1;
```

TAG_INTERNET_IOP

```
const ProfileId TAG_INTERNET_IOP = 0;
```

TAG_MULTIPLE_COMPONENTS

```
const ProfileId TAG_MULTIPLE_COMPONENTS = 1;
```

TransactionService

```
const ServiceId TransactionService = 0;
```

Structs

IOR

```
struct IOR
{
    string type_id;
    sequence<TaggedProfile> profiles;
};
```

ServiceContext

```
struct ServiceContext
{
    ServiceId context_id;
    sequence<octet> context_data;
};
```

TaggedComponent

```
struct TaggedComponent
{
    ComponentId tag;
    sequence<octet> component_data;
};
```

TaggedProfile

```
struct TaggedProfile
{
    ProfileId tag;
    sequence<octet> profile_data;
};
```

Aliases

ComponentId

```
typedef unsigned long ComponentId;
```

MultipleComponentProfile

```
typedef sequence<TaggedComponent> MultipleComponentProfile;
```

ProfileId

```
typedef unsigned long ProfileId;
```

ServiceContextList

```
typedef sequence<ServiceContext> ServiceContextList;
```

ServiceId

```
typedef unsigned long ServiceId;
```

Module OB

Interface Index

ConnectionReusePolicy

The connection reuse policy.

ProtocolPolicy

The protocol policy.

ReconnectPolicy

The reconnect policy.

TimeoutPolicy

The timeout policy.

Constants

CONNECTION_REUSE_POLICY

```
const CORBA::PolicyType CONNECTION_REUSE_POLICY = 3;
```

This policy type identifies the connection reuse policy.

PROTOCOL_POLICY

```
const CORBA::PolicyType PROTOCOL_POLICY = 2;
```

This policy type identifies the protocol policy.

RECONNECT_POLICY

```
const CORBA::PolicyType RECONNECT_POLICY = 4;
```

This policy type identifies the reconnect policy.

TIMEOUT_POLICY

```
const CORBA::PolicyType TIMEOUT_POLICY = 5;
```

This policy type identifies the timeout policy.

FSSL Bibliography

[1] The SSL Protocol, Version 3.0, Transport Layer Security Working Group.

[2] ANSI X3.106, American National Standard for Information Systems-Data Link Encryption, American National Standards Institute, 1983.

[3] R. Rivest, A. Shamir, and L. M. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, Communications of the ACM, v. 21, n. 2, Feb 1978, pp. 120-126.

[4] CCITT. Recommendation X.509: The Directory - Authentication Framework. 1988.

[5] SSLeay and SSLapps FAQ, T. J. Hudson, E. A. Young.

[6] iSaSiLk 2.0 User Manual, Institute for Applied Information Processing and Communications, Graz University of Technology, 1998.

[7] NIST FIPS PUB 180-1, Secure Hash Standard, National Institute of Standards and Technology, U.S. Department of Commerce, DRAFT, 31 May 1994.

[8] R. Rivest. RFC 1321: The MD5 Message Digest Algorithm, April 1992.

[9] W. Diffie and M. E. Hellman, New Directions in Cryptography, IEEE Transactions on Information Theory, V.IT-22, n. 6, Jun 1977, pp. 74-84.

[10] Marc Laukien, Uwe Seimet, Matthew Newhook, and Mark Spruiell, ORBacus For C++ and Java, Object Oriented Concepts, Inc.

[11] Bruce Schneier, Applied Cryptography, John Wiley & Sons, Inc.

[12] PUB 46-1 National Bureau of Standards. FIPS PUB 46-1: Data Encryption Standard. January 1988.

[13] CCITT. Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1). 1988.

[14] RFC 1421 Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures," RFC 1421 February 1993.

[15] RFC 1422 Kent, S., "Privacy Enhancement for Internet Electronic Mail: Part II: Certificate- Based Key Management," RFC 1422, February 1993.

[16] RFC 1423 Balenson, D., "Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers," RFC 1423, February 1993.

[17] RFC 1424 Kaliski, B., "Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services," RFC 1424, February 1993.

[18] PKCS #8: Private-Key Information Syntax Standard, An RSA Laboratories Technical Note, Version 1.2, Revised November 1, 1993.