

Table of Contents

javax.net and javax.net.ssl.....	1
Package javax.net.....	1
ServerSocketFactory.....	1
SocketFactory.....	2
Package javax.net.ssl.....	3
CertPathTrustManagerParameters.....	5
HandshakeCompletedEvent.....	5
HandshakeCompletedListener.....	6
HostnameVerifier.....	7
HttpsURLConnection.....	8
KeyManager.....	9
KeyManagerFactory.....	9
KeyManagerFactorySpi.....	10
KeyStoreBuilderParameters.....	11
ManagerFactoryParameters.....	11
SSLContext.....	12
SSLContextSpi.....	13
SSLEngine.....	13
SSLEngineResult.....	15
SSLEngineResult.HandshakeStatus.....	16
SSLEngineResult.Status.....	17
SSLException.....	17
SSLHandshakeException.....	18
SSLKeyException.....	18
SSLPeerUnverifiedException.....	18
SSLPermission.....	19
SSLProtocolException.....	20
SSLServerSocket.....	20
SSLServerSocketFactory.....	21
SSLSession.....	22
SSLSessionBindingEvent.....	24
SSLSessionBindingListener.....	24
SSLSessionContext.....	25
SSLSocket.....	26
SSLSocketFactory.....	28
TrustManager.....	29
TrustManagerFactory.....	29
TrustManagerFactorySpi.....	30
X509ExtendedKeyManager.....	31
X509KeyManager.....	31
X509TrustManager.....	32

Chapter 18. javax.net and javax.net.ssl

This chapter documents the `javax.net` package and, more importantly, its subpackage `javax.net.ssl`. These packages were originally defined by the Java Secure Sockets Extension (JSSE) before they were integrated into Java 1.4, which is why they have a "javax" prefix.

`javax.net` is a small package that simply defines abstract factory classes for creating network sockets and servers sockets. `javax.net.ssl` provides subclasses of these factory classes that have the specific purpose of creating sockets and server sockets that enable secure network communication through the SSL protocol and the closely-related TLS protocol.

Package javax.net

Java 1.4

This small package defines factory classes for creating sockets and server sockets. These factory classes can be used to create regular `java.net.Socket` and `java.net.ServerSocket` objects. More importantly, however, these factory classes can be subclassed to serve as factories for other types of sockets such as the SSL-enabled sockets of the `javax.net.ssl` package.

Classes

```
public abstract class ServerSocketFactory;  
public abstract class SocketFactory;
```

ServerSocketFactory

javax.net

Java 1.4

This abstract class defines a factory API for creating server socket objects. Use the static `getDefault()` method to obtain a default `ServerSocketFactory` object that is suitable for creating regular `java.net.ServerSocket` sockets. Once you have a `ServerSocketFactory` object, call one of the `createServerSocket()` methods to create a new socket and optionally bind it to a local port and specify the allowed backlog

of queued connections. See `javax.net.ssl.SSLServerSocketFactory` for a socket factory that can create secure `javax.net.ssl.SSLServerSocket` objects.

```
public abstract class ServerSocketFactory {
    // Protected Constructors
    protected ServerSocketFactory( );
    // Public Class Methods
    public static ServerSocketFactory getDefault( );
    // Public Instance Methods
    public java.net.ServerSocket createServerSocket( ) throws java.io.IOException;
    public abstract java.net.ServerSocket createServerSocket(int port)
        throws java.io.IOException;
    public abstract java.net.ServerSocket createServerSocket(int port,
        int backlog) throws java.io.IOException;
    public abstract java.net.ServerSocket createServerSocket(int port,
        int backlog, java.net.InetAddress ifAddress) throws java.io.IOException;
}
```

Subclasses

`javax.net.ssl.SSLServerSocketFactory`

Returned By

`javax.net.ssl.SSLServerSocketFactory.getDefault()`

SocketFactory

javax.net

Java 1.4

This abstract class defines a factory API for creating socket objects. Use the static `getDefault()` method to obtain a default `SocketFactory` object that is suitable for creating regular `java.net.Socket` sockets. (This default `SocketFactory` is the one used by the `Socket()` constructor, which usually provides an easier way to create normal sockets.) Once you have a `SocketFactory` object, call one of the `createSocket()` methods to create a new socket and optionally connect it to a remote host and optionally bind it to a local address and port. See `javax.net.ssl.SSLSocketFactory` for a socket factory that can create secure `javax.net.ssl.SSLSocket` objects.

```
public abstract class SocketFactory {
    // Protected Constructors
    protected SocketFactory( );
    // Public Class Methods
    public static SocketFactory getDefault( );
    // Public Instance Methods
    public java.net.Socket createSocket( ) throws java.io.IOException;
    public abstract java.net.Socket createSocket(String host, int port)
        throws java.io.IOException, java.net.UnknownHostException;
    public abstract java.net.Socket createSocket(java.net.InetAddress host,
        int port) throws java.io.IOException;
    public abstract java.net.Socket createSocket(java.net.InetAddress address,
        int port, java.net.InetAddress localAddress,
        int localPort)
        throws java.io.IOException;
    public abstract java.net.Socket createSocket(String host, int port,
        java.net.InetAddress localHost, int localPort)
        throws java.io.IOException, java.net.UnknownHostException;
}
```

Chapter 18. javax.net and javax.net.ssl

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Prepared for Ronald Fischer, Safari ID: ronald.fischer@fussuhhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.

This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

Subclasses`javax.net.ssl.SSLSocketFactory`**Returned By**`javax.net.ssl.SSLSocketFactory.getDefault()`**Package javax.net.ssl****Java 1.4**

This package defines an API for secure network sockets using the SSL (Secure Sockets Layer) protocol, or the closely related TLS (Transport Layer Security) protocol. It defines the `SSLSocket` and `SSLServerSocket` subclasses of the `java.net.Socket` and `java.net.ServerSocket` classes. And it defines `SSLSocketFactory` and `SSLServerSocketFactory` subclasses of the `javax.net.SocketFactory` classes to create those SSL-enabled sockets and server sockets. Clients that want to perform simple SSL-enabled networking can create an `SSLSocket` with code like the following:

```
SSLSocketFactory factory = SSLSocketFactory.getDefault( );
SSLSocket securesock = (SSLSocket)factory.getSocket(hostname,
                                                    443);    // https port
```

Once an `SSLSocket` has been created, it can be used just like a normal `java.net.Socket`. Once a connection is established over an `SSLSocket`, you can use the `getSession()` method to obtain an `SSLSession` object that provides information about the connection. Note that despite the name of this package and of its key classes, it supports the TLS protocol in addition to the SSL. (The default provider in Sun's implementation supports SSL 3.0 and TLS 1.0.) The TLS protocol is closely related to SSL, and we'll simply use the term SSL here.

The `SSLSocket` class allows you to do arbitrary networking with an SSL-enabled peer. The most common use of SSL today is with the `https:` protocol on the web. The addition of this package to the core Java platform enables support for `https:` URLs in the `java.net.URL` class, which allows you to securely transfer data over the web without having to directly use this package at all. When you call `openConnection()` on a `https:` URL, the `URLConnection` object that is returned can be cast to an `HttpsURLConnection` object, which defines some SSL-specific methods. See `java.net.URL` and `java.net.URLConnection` for more information about networking with URLs.

Although the code shown above to create a `SSLSocket` is quite simple, this package is much more complex because it exposes a lot of SSL infrastructure so that applications with advanced networking needs can configure it as needed. Also, like all security-related packages, this one is provider-based and algorithm-independent, which adds a layer of complexity. If you want to explore this package beyond the two socket classes, the two factory classes, and the `HttpsURLConnection` class, start with `SSLContext`. This class is a factory for socket factories, and as such is the central class of the API. To customize the way SSL networking is done, you create an `SSLContext` optionally specifying the desired provider of the implementation. Next, you initialize the `SSLContext` by providing a custom `KeyManager` as a source of authentication information to be supplied to the remote host if required, a custom `TrustManager` as a verifier for the authentication information (if any) presented by the remote host, and a custom `java.security.SecureRandom` object as a source of randomness. Once the `SSLContext` is initialized in this way, you can use it to create `SSLSocketFactory` and `SSLServerSocketFactory` objects that use the `KeyManager` and `TrustManager` objects you supplied.

In Java 5.0, the `SSLContext` can also be used to create an `SSLEngine` object, which performs transport-independent SSL encryption of outbound packets and SSL decryption of inbound packets. This enables the use of SSL with the nonblocking networking facilities of the `java.nio.channels` package, for example.

Interfaces

```
public interface HandshakeCompletedListener extends java.util.EventListener;
public interface HostnameVerifier;
public interface KeyManager;
public interface ManagerFactoryParameters;
public interface SSLSession;
public interface SSLSessionBindingListener extends java.util.EventListener;
public interface SSLSessionContext;
public interface TrustManager;
public interface X509KeyManager extends KeyManager;
public interface X509TrustManager extends TrustManager;
```

Enumerated Types

```
public enum SSLEngineResult.HandshakeStatus;
public enum SSLEngineResult.Status;
```

Events

```
public class HandshakeCompletedEvent extends java.util.EventObject;
public class SSLSessionBindingEvent extends java.util.EventObject;
```

Other Classes

```
public class CertPathTrustManagerParameters implements ManagerFactoryParameters;
public abstract class HttpsURLConnection extends java.net.HttpURLConnection;
public class KeyManagerFactory;
public abstract class KeyManagerFactorySpi;
public class KeyStoreBuilderParameters implements ManagerFactoryParameters;
public class SSLContext;
public abstract class SSLContextSpi;
public abstract class SSLEngine;
public class SSLEngineResult;
```

Chapter 18. javax.net and javax.net.ssl

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Prepared for Ronald Fischer, Safari ID: ronald.fischer@fussuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.

This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

```

public final class SSLPermission extends java.security.BasicPermission;
public abstract class SSLServerSocket extends java.net.ServerSocket;
public abstract class SSLServerSocketFactory extends javax.net.ServerSocketFactory;
public abstract class SSLSocket extends java.net.Socket;
public abstract class SSLSocketFactory extends javax.net.SocketFactory;
public class TrustManagerFactory;
public abstract class TrustManagerFactorySpi;
public abstract class X509ExtendedKeyManager implements X509KeyManager;

```

Exceptions

```

public class SSLException extends java.io.IOException;
public class SSLHandshakeException extends SSLException;
public class SSLKeyException extends SSLException;
public class SSLPeerUnverifiedException extends SSLException;
public class SSLProtocolException extends SSLException;

```

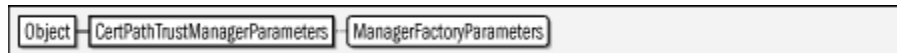
CertPathTrustManagerParameters

javax.net.ssl

Java 5.0

This class implements the `ManagerFactoryParameters` interface and wraps a `java.security.cert.CertPathParameters` object used to initialize a `TrustManager` based on a certificate path. See the `init()` method of `TrustManagerFactory`.

Figure 18-1. javax.net.ssl.CertPathTrustManagerParameters



```

public class CertPathTrustManagerParameters implements ManagerFactoryParameters {
// Public Constructors
    public CertPathTrustManagerParameters(java.security.cert.CertPathParameters parameters);
// Public Instance Methods
    public java.security.cert.CertPathParameters getParameters();
}

```

HandshakeCompletedEvent

javax.net.ssl

Java 1.4

serializable event

An instance of this class is passed to the `handshakeCompleted()` method of any registered `HandshakeCompletedListener` objects by an `SSLSocket` when that socket completes the handshake phase of establishing a connection. The various methods of a `HandshakeCompletedEvent` return information (such as the name of the cipher suite

in use and the certificate chain of the remote host) that was determined during that handshake.

Note that the `getPeerCertificateChain()` method returns an object from the `javax.security.cert` package, which is not documented in this book. The method and package exist only for backward compatibility with earlier versions of the JSSE API, and should be considered deprecated. Use `getPeerCertificates()`, which uses `java.security.cert` instead.

Figure 18-2. javax.net.ssl.HandshakeCompletedEvent



```

public class HandshakeCompletedEvent extends java.util.EventObject {
    // Public Constructors
    public HandshakeCompletedEvent(SSLSocket sock, SSLSession s);
    // Public Instance Methods
    public String getCipherSuite();
    public java.security.cert.Certificate[] getLocalCertificates();
    5.0 public java.security.Principal getLocalPrincipal();
    public javax.security.cert.X509Certificate[] getPeerCertificateChain()
        throws SSLPeerUnverifiedException;
    public java.security.cert.Certificate[] getPeerCertificates()
        throws SSLPeerUnverifiedException;
    5.0 public java.security.Principal getPeerPrincipal() throws SSLPeerUnverifiedException;
    public SSLSession getSession();
    public SSLSocket getSocket();
}
  
```

Passed To

`HandshakeCompletedListener.handshakeCompleted()`

HandshakeCompletedListener

javax.net.ssl

Java 1.4

event listener

This interface is implemented by any class that wants to receive notifications (in the form of a call to `handshakeCompleted()` method) when an `SSLSocket` completes the SSL handshake. Register a `HandshakeCompletedListener` for an `SSLSocket` by passing it to the `addHandshakeCompletedListener()` method of the socket. When the socket completes the handshake phase of connection, it will call the `handshakeCompleted()` method of all registered listeners, passing in a `HandshakeCompletedEvent` object.

Figure 18-3. javax.net.ssl.HandshakeCompletedListener

```

public interface HandshakeCompletedListener extends java.util.EventListener {
    // Public Instance Methods
    void handshakeCompleted(HandshakeCompletedEvent event);
}

```

Passed To

```

SSLSocket.{addHandshakeCompletedListener( ),
removeHandshakeCompletedListener( )}

```

HostnameVerifier**javax.net.ssl****Java 1.4**

An object that implements this interface may be used with an `HttpsURLConnection` object to handle the case in which the hostname that appears in the URL does not match the hostname obtained during the SSL handshake with the server. This occurs, for example, when a website uses the secure certificate of its parent web hosting company, for example. In this situation, the `verify()` method of the `HostnameVerifier` is called to determine whether the connection should proceed or not. `verify()` should return `true` to allow the connection to proceed, and should return `false` to cause the connection to fail. The `hostname` argument to `verify()` specifies the hostname that appeared in the URL. The `session` argument specifies the `SSLSession` object that was established during the handshake. Call `getPeerHost()` on this object to determine the hostname reported during server authentication. If no `HostnameVerifier` is registered with a `HttpsURLConnection` object, and no default verifier is registered with the `HttpsURLConnection` class, then hostname mismatches will always cause the connection to fail. In user-driven applications such as web browsers, a `HostnameVerifier` can be used to ask the user whether to proceed or not.

```

public interface HostnameVerifier {
    // Public Instance Methods
    boolean verify(String hostname, SSLSession session);
}

```

Passed To

```

HttpsURLConnection.{setDefaultHostnameVerifier( ),
setHostnameVerifier( )}

```

Returned By

```

HttpsURLConnection.{getDefaultHostnameVerifier( ),
getHostnameVerifier( )}

```

Chapter 18. javax.net and javax.net.ssl

Type Of

HttpsURLConnection.hostnameVerifier

HttpsURLConnection**javax.net.ssl****Java 1.4**

This class is a `java.net.URLConnection` for a URL that uses the `https:` protocol. It extends `java.net.HttpURLConnection` and, in addition to inheriting the methods of its superclasses, it defines methods for specifying the `SSLSocketFactory` and `HostnameVerifier` to use when establishing the connection. Static versions of these methods allow you to specify a default factory and verifier objects for use with all `HttpsURLConnection` objects. After the connection has been established, several other methods exist to obtain information (such as the cipher suite and the server certificates) about the connection itself.

Obtain a `HttpsURLConnection` object by calling the `openConnection()` method of a URL that uses the `https://` protocol specifier, and casting the returned value to this type. The `HttpsURLConnection` object is unconnected at this point, and you can call `setHostnameVerifier()` and `setSSLSocketFactory()` to customize the way the connection is made. (If you do not specify a `HostnameVerifier` for the instance, or a default one for the class, then hostname mismatches will always cause the connection to fail. If you do not specify an `SSLSocketFactory` for the instance or class, then a default one will be used.) To connect, call the inherited `connect()` method, and then call the inherited `getContent()` to retrieve the content of the URL as an object, or use the inherited `getInputStream()` to obtain a `java.io.InputStream` with which you can read the content of the URL.

Figure 18-4. javax.net.ssl.HttpsURLConnection

```

public abstract class HttpsURLConnection extends java.net.HttpURLConnection {
// Protected Constructors
    protected HttpsURLConnection(java.net.URL url);
// Public Class Methods
    public static HostnameVerifier getDefaultHostnameVerifier();
    public static SSLSocketFactory getDefaultSSLSocketFactory();
    public static void setDefaultHostnameVerifier(HostnameVerifier v);
    public static void setDefaultSSLSocketFactory(SSLSocketFactory sf);
// Public Instance Methods
    public abstract String getCipherSuite();
    public HostnameVerifier getHostnameVerifier();
    public abstract java.security.cert.Certificate[] getLocalCertificates();
5.0 public java.security.Principal getLocalPrincipal();
5.0 public java.security.Principal getPeerPrincipal()
    throws SSLPeerUnverifiedException;
  
```

Chapter 18. javax.net and javax.net.ssl

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
 Print Publication Date: 2005/03/01

Prepared for Ronald Fischer, Safari ID: ronald.fischer@fussshuhn.de
 User number: 628024 Copyright 2008, Safari Books Online, LLC.

This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

```

    public abstract java.security.cert.Certificate[ ] getServerCertificates( )
        throws SSLPeerUnverifiedException;
    public SSLSocketFactory getSSLSocketFactory( );
    public void setHostnameVerifier(HostnameVerifier v);
    public void setSSLSocketFactory(SSLSocketFactory sf);
    // Protected Instance Fields
    protected HostnameVerifier hostnameVerifier;
}

```

KeyManager**javax.net.ssl****Java 1.4**

This is a marker interface to identify key manager objects. A key manager is responsible for obtaining and managing authentication credentials (such as a certificate chain and an associated private key) that the local host can use to authenticate itself to the remote host. It is usually used on the server-side of an SSL connection, but can be used on the client-side as well.

Use a `KeyManagerFactory` to obtain `KeyManager` objects. `KeyManager` objects returned by a `KeyManagerFactory` can always be cast to a subinterface specific to a particular type of authentication credentials. See `X509KeyManager`, for example.

```

public interface KeyManager {
}

```

Implementations

`X509KeyManager`

Passed To

`SSLContext.init(), SSLContextSpi.engineInit()`

Returned By

`KeyManagerFactory.getKeyManagers(),`
`KeyManagerFactorySpi.engineGetKeyManagers()`

KeyManagerFactory**javax.net.ssl****Java 1.4**

A `KeyManagerFactory` is responsible for creating `KeyManager` objects for a specific key management algorithm. Obtain a `KeyManagerFactory` object by calling one of the `getInstance()` methods and specifying the desired algorithm and, optionally, the desired provider. In Java 1.4, the "SunX509" algorithm is the only one supported by the

default "SunJSSE" provider. After calling `getInstance()`, you initialize the factory object with `init()`. For the "SunX509" algorithm, you always use the two-argument version of `init()` passing in a `KeyStore` object that contains the private keys and certificates required by `X509KeyManager` objects, and also specifying the password used to protect the private keys in that `KeyStore`. Once a `KeyManagerFactory` has been created and initialized, use it to create a `KeyManager` by calling `getKeyManagers()`. This method returns an array of `KeyManager` objects because some key management algorithms may handle more than one type of key. The "SunX509" algorithm manages only X509 keys, and always returns an array with an `X509KeyManager` object as its single element. This returned array is typically passed to the `init()` method of an `SSLContext` object.

If a `KeyStore` and password are not passed to the `init()` method of the `KeyManagerFactory` for the "SunX509" algorithm, then the factory uses attempts to read a `KeyStore` from the file specified by the `javax.net.ssl.keyStore` system property using the password specified by the `javax.net.ssl.keyStorePassword`. The type of the keystore is specified by `javax.net.ssl.keyStoreType`.

```
public class KeyManagerFactory {
// Protected Constructors
    protected KeyManagerFactory(KeyManagerFactorySpi factorySpi,
        java.security.Provider provider, String algorithm);
// Public Class Methods
    public static final String getDefaultAlgorithm( );
    public static final KeyManagerFactory getInstance(String algorithm)
        throws java.security.NoSuchAlgorithmException;
    public static final KeyManagerFactory getInstance(String algorithm,
        java.security.Provider provider)
        throws java.security.NoSuchAlgorithmException;
    public static final KeyManagerFactory getInstance(String algorithm,
        String provider) throws java.security.NoSuchAlgorithmException,
        java.security.NoSuchProviderException;
// Public Instance Methods
    public final String getAlgorithm( );
    public final KeyManager[ ] getKeyManagers( );
    public final java.security.Provider getProvider( );
    public final void init(ManagerFactoryParameters spec)
        throws java.security.InvalidAlgorithmParameterException;
    public final void init(java.security.KeyStore ks, char[ ] password)
        throws java.security.KeyStoreException,
        java.security.NoSuchAlgorithmException,
        java.security.UnrecoverableKeyException;
}
```

KeyManagerFactorySpi

javax.net.ssl

Java 1.4

This abstract class defines the Service Provider Interface for `KeyManagerFactory`. Security providers must implement this interface, but applications never need to use it.

```
public abstract class KeyManagerFactorySpi {
    // Public Constructors
    public KeyManagerFactorySpi( );
    // Protected Instance Methods
    protected abstract KeyManager[] engineGetKeyManagers( );
    protected abstract void engineInit(ManagerFactoryParameters spec)
        throws java.security.InvalidAlgorithmParameterException;
    protected abstract void engineInit(java.security.KeyStore ks, char[] password)
        throws java.security.KeyStoreException, java.security.NoSuchAlgorithmException,
            java.security.UnrecoverableKeyException;
}
```

Passed To

`KeyManagerFactory.KeyManagerFactory()`

KeyStoreBuilderParameters

javax.net.ssl

Java 5.0

This class implements the `ManagerFactoryParameters` interface and encapsulates a `java.util.List` of `java.security.KeyStore.Builder` object for use by an `X509KeyManager`. See the `init()` method of `KeyManagerFactory`.

Figure 18-5. javax.net.ssl.KeyStoreBuilderParameters



```
public class KeyStoreBuilderParameters implements ManagerFactoryParameters {
    // Public Constructors
    public KeyStoreBuilderParameters(java.util.List parameters);
    public KeyStoreBuilderParameters(java.security.KeyStore.Builder builder);
    // Public Instance Methods
    public java.util.List getParameters( );
}
```

ManagerFactoryParameters

javax.net.ssl

Java 1.4

This marker interface identifies objects that provide algorithm-specific or provider-specific initialization parameters for `KeyManagerFactory` and `TrustManagerFactory` objects. In the default "SunJSSE" provider shipped by Sun, the

only supported type for these factory classes is "SunX509". Factories of these types need to be initialized with a `KeyStore` object but do not require any specialized `ManagerFactoryParameters` object. Therefore, the `javax.net.ssl` package does not define any subinterfaces of this interface, and it is never used with the default provider. Third-party or future providers may use it, however.

```
public interface ManagerFactoryParameters {
}
```

Implementations

`CertPathTrustManagerParameters`, `KeyStoreBuilderParameters`

Passed To

```
KeyManagerFactory.init( ), KeyManagerFactorySpi.engineInit( ),
TrustManagerFactory.init( ), TrustManagerFactorySpi.engineInit( )
```

SSLContext

javax.net.ssl

Java 1.4

This class is a factory for socket and server socket factories. Although most applications do not need to use this class directly, it is the central class of the `javax.net.ssl` package. Most applications use the default `SSLContext` and `SSLServerSocketFactory` objects returned by the static `getDefault()` methods of those classes. Applications that want to perform SSL networking using a security provider other than the default provider, or that want to customize key management or trust management for the SSL connection should use custom socket factories created from a custom `SSLContext`. In Java 5.0, this class also includes `createSSLEngine()` factory methods for creating `SSLEngine` objects.

Create an `SSLContext` by passing the name of the desired secure socket protocol and, optionally, the desired provider to `getInstance()`. The default "SunJSSE" provider supports protocol strings "SSL", "SSLv2", "SSLv3", "TLS", and "TLSv1". Once you have created an `SSLContext` object, call its `init()` method to supply the `KeyManager`, `TrustManager`, and `SecureRandom` objects it requires. If any of the `init()` arguments is null, a default value will be used. Finally, obtain a `SSLContext` and `SSLServerSocketFactory` by calling `getSocketFactory()` and `getServerSocketFactory()`.

```
public class SSLContext {
    // Protected Constructors
    protected SSLContext(SSLContextSpi contextSpi, java.security.Provider provider,
        String protocol);
    // Public Class Methods
    public static SSLContext getInstance(String protocol)
```

Chapter 18. javax.net and javax.net.ssl

```

        throws java.security.NoSuchAlgorithmException;
    public static SSLContext getInstance(String protocol, String provider)
        throws java.security.NoSuchAlgorithmException,
            java.security.NoSuchProviderException;
    public static SSLContext getInstance(String protocol, java.security.Provider provider)
        throws java.security.NoSuchAlgorithmException;
    // Public Instance Methods
    5.0 public final SSLEngine createSSLEngine();
    5.0 public final SSLEngine createSSLEngine(String peerHost, int peerPort);
    public final SSLSessionContext getClientSessionContext();
    public final String getProtocol();
    public final java.security.Provider getProvider();
    public final SSLSessionContext getServerSessionContext();
    public final SSLServerSocketFactory getServerSocketFactory();
    public final SSLSocketFactory getSocketFactory();
    public final void init(KeyManager[] km, TrustManager[] tm,
        java.security.SecureRandom random)
        throws java.security.KeyManagementException;
}

```

SSLContextSpi**javax.net.ssl****Java 1.4**

This abstract class defines the Service Provider Interface for `SSLContext`. Security providers must implement this interface, but applications never need to use it.

```

public abstract class SSLContextSpi {
    // Public Constructors
    public SSLContextSpi();
    // Protected Instance Methods
    5.0 protected abstract SSLEngine engineCreateSSLEngine();
    5.0 protected abstract SSLEngine engineCreateSSLEngine(String host, int port);
    protected abstract SSLSessionContext engineGetClientSessionContext();
    protected abstract SSLSessionContext engineGetServerSessionContext();
    protected abstract SSLServerSocketFactory engineGetServerSocketFactory();
    protected abstract SSLSocketFactory engineGetSocketFactory();
    protected abstract void engineInit(KeyManager[] km, TrustManager[] tm,
        java.security.SecureRandom sr)
        throws java.security.KeyManagementException;
}

```

Passed To

`SSLContext.SSLContext()`

SSLEngine**javax.net.ssl****Java 5.0**

This class performs SSL handshaking, encryption and decryption, but does not send or receive messages over the network. This leaves the network transport mechanism up to

the user of this class, and enables SSL communication using the nonblocking I/O mechanisms of the `java.nio` package. The price of this flexibility is that your code must follow a relatively complex protocol to use an `SSLEngine` correctly.

Create an `SSLEngine` with `SSLContext.createSSLEngine()`. Next, configure it with the various setter methods to specify authentication requirements, encryption algorithms, etc. After creating and configuring an engine, you use it to encrypt outbound data from one `ByteBuffer` to another with `wrap()` and to decrypt inbound data from one byte buffer to another with `unwrap()`. (Note that the `wrap()` and `unwrap()` methods also come in gathering and scattering variants.) Both methods return an `SSLEngineResult`.

The initial call or calls to `wrap()` produce outbound handshaking data without consuming any of the source bytes in the buffer you provide. Initial calls to `unwrap()` may consume inbound handshaking data without producing any result bytes. Monitor the `SSLEngineResult.HandshakeStatus` value to ensure that handshaking is proceeding as needed. When handshaking is complete, you can call `getSession()` to obtain the `SSLSession` object that describes session details negotiated during handshaking. Remember that either peer of an SSL connection may request a new handshake at any time; this means that you must monitor the `HandshakeStatus` after every `wrap()` or `unwrap()` call in case a new handshake has been requested. You can request a new handshake yourself with `beginHandshake()`.

As part of the handshaking protocol, the `SSLEngine` typically needs to use the `KeyManager` or `TrustManager` of the originating `SSLContext` object. Rather than blocking a `wrap()` or `unwrap()` method while these operations are performed, it instead returns an `SSLResult.HandshakeStatus`, indicating that a task needs to be performed. When this happens, you must call `getDelegatedTask()` repeatedly, calling the `run()` methods of the `Runnable` objects it returns until it returns `null` to indicate that all necessary tasks have been completed. (If it returns more than one `Runnable`, it is safe to run them in parallel (with a `java.util.concurrent.ExecutorCompletionService`, for example). Once all such tasks have been run, the original call to `wrap()` or `unwrap()` should be repeated.

When you are done sending outbound data, call `closeOutbound()`, and then call `wrap()` one or more times to flush any remaining data from the engine. Call `wrap()` until the returned `SSLEngineResult.Status` indicates that the connection has closed. Similarly, if you are done reading inbound data, call `closeInbound()` and final calls to `unwrap()` until the connection is closed.

It is safe for one thread to call `wrap()` while another thread is calling `unwrap()`. It is not safe, however, for either method to be called by two threads at once.

```
public abstract class SSLEngine {
// Protected Constructors
    protected SSLEngine( );
    protected SSLEngine(String peerHost, int peerPort);
// Public Instance Methods
    public abstract void beginHandshake( ) throws SSLException;
    public abstract void closeInbound( ) throws SSLException;
    public abstract void closeOutbound( );
    public abstract Runnable getDelegatedTask( );
    public abstract String[ ] getEnabledCipherSuites( );
    public abstract String[ ] getEnabledProtocols( );
    public abstract boolean getEnableSessionCreation( );
    public abstract SSLEngineResult.HandshakeStatus getHandshakeStatus( );
    public abstract boolean getNeedClientAuth( );
    public String getPeerHost( );
    public int getPeerPort( );
    public abstract SSLSession getSession( );
    public abstract String[ ] getSupportedCipherSuites( );
    public abstract String[ ] getSupportedProtocols( );
    public abstract boolean getUseClientMode( );
    public abstract boolean getWantClientAuth( );
    public abstract boolean isInboundDone( );
    public abstract boolean isOutboundDone( );
    public abstract void setEnabledCipherSuites(String[ ] suites);
    public abstract void setEnabledProtocols(String[ ] protocols);
    public abstract void setEnableSessionCreation(boolean flag);
    public abstract void setNeedClientAuth(boolean need);
    public abstract void setUseClientMode(boolean mode);
    public abstract void setWantClientAuth(boolean want);
    public SSLEngineResult unwrap(java.nio.ByteBuffer src, java.nio.
        ByteBuffer dst) throws SSLException;
    public SSLEngineResult unwrap(java.nio.ByteBuffer src, java.nio.
        ByteBuffer[ ] dsts) throws SSLException;
    public abstract SSLEngineResult unwrap(java.nio.ByteBuffer src,
        java.nio.ByteBuffer[ ] dsts, int offset,
        int length) throws SSLException;
    public SSLEngineResult wrap(java.nio.ByteBuffer[ ] srcs, java.nio.
        ByteBuffer dst) throws SSLException;
    public SSLEngineResult wrap(java.nio.ByteBuffer src, java.nio.
        ByteBuffer dst) throws SSLException;
    public abstract SSLEngineResult wrap(java.nio.ByteBuffer[ ] srcs,
        int offset, int length,
        java.nio.ByteBuffer dst) throws SSLException;
}
```

Passed To

```
X509ExtendedKeyManager.{chooseEngineClientAlias( ),
chooseEngineServerAlias( )}
```

Returned By

```
SSLContext.createSSLEngine( ),
SSLContextSpi.engineCreateSSLEngine( )
```

SSLEngineResult

javax.net.ssl

Java 5.0

Chapter 18. javax.net and javax.net.ssl

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Prepared for Ronald Fischer, Safari ID: ronald.fischer@fussshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.

This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

An object of this type is returned by the `wrap()` and `unwrap()` methods of an `SSLEngine`. Use the methods of this object to determine how much data was consumed and produced and to obtain the `Status` of the operation and the `HandshakeStatus` of the connection. These two nested enumerated types return important values. Correct operation of an `SSLEngine` requires that your code respond correctly to the `Status` and `HandshakeStatus` results.

```
public class SSLEngineResult {
    // Public Constructors
    public SSLEngineResult(SSLEngineResult.Status status, SSLEngineResult.
        HandshakeStatus handshakeStatus,
        int bytesConsumed, int bytesProduced);
    // Nested Types
    public enum HandshakeStatus;
    public enum Status;
    // Public Instance Methods
    public final int bytesConsumed( );
    public final int bytesProduced( );
    public final SSLEngineResult.HandshakeStatus getHandshakeStatus( );
    public final SSLEngineResult.Status getStatus( );
    // Public Methods Overriding Object
    public String toString( );
}
```

Returned By

`SSLEngine.{unwrap(), wrap()}`

SSLEngineResult.HandshakeStatus

javax.net.ssl

Java 5.0

serializable comparable enum

The constants defined by this enumerated type specify the handshake status of the `SSLEngine` and often specify the action your code must take next in order to ensure correct operation. The values are the following:

```
public enum SSLEngineResult.HandshakeStatus {
    // Enumerated Constants
    NOT_HANDSHAKING,
    FINISHED,
    NEED_TASK,
    NEED_WRAP,
    NEED_UNWRAP;
    // Public Class Methods
    public static SSLEngineResult.HandshakeStatus valueOf(String name);
    public static final SSLEngineResult.HandshakeStatus[] values( );
}
```

Passed To

`SSLEngineResult.SSLEngineResult()`

Returned By

`SSLEngine.getHandshakeStatus(),`
`SSLEngineResult.getHandshakeStatus()`

Chapter 18. javax.net and javax.net.ssl

SSLEngineResult.Status**javax.net.ssl****Java 5.0*****serializable comparable enum***

The constants of this enumerated type indicate the status of a `wrap()` or `unwrap()` operation:

```
public enum SSLEngineResult.Status {
    // Enumerated Constants
    BUFFER_UNDERFLOW,
    BUFFER_OVERFLOW,
    OK,
    CLOSED;
    // Public Class Methods
    public static SSLEngineResult.Status valueOf(String name);
    public static final SSLEngineResult.Status[] values();
}
```

Passed To

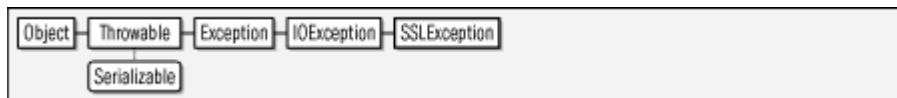
```
SSLEngineResult.SSLEngineResult()
```

Returned By

```
SSLEngineResult.getStatus()
```

SSLException**javax.net.ssl****Java 1.4*****serializable checked***

Signals an SSL-related problem. This class serves as the common superclass of more specific SSL exception subclasses.

Figure 18-6. javax.net.ssl.SSLException

```
public class SSLException extends java.io.IOException {
    // Public Constructors
    5.0 public SSLException(Throwable cause);
    public SSLException(String reason);
    5.0 public SSLException(String message, Throwable cause);
}
```

Subclasses

`SSLHandshakeException`, `SSLKeyException`, `SSLPeerUnverifiedException`, `SSLProtocolException`

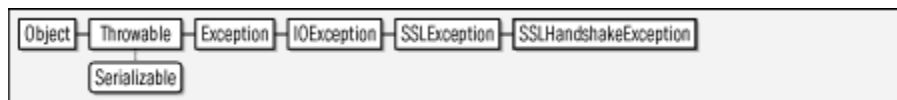
Chapter 18. javax.net and javax.net.ssl

Thrown By

```
SSLEngine.{beginHandshake( ),closeInbound( ),unwrap( ),wrap( )}
```

SSLHandshakeException**javax.net.ssl****Java 1.4*****serializable checked***

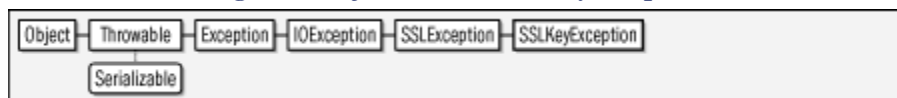
Signals that the SSL handshake failed for some reason other than failed authentication (see `SSLPeerUnverifiedException`). For example, it may be thrown because the client and server cannot agree on a mutually-acceptable cipher suite. When this exception is thrown, the `SSLSocket` object is no longer usable.

Figure 18-7. javax.net.ssl.SSLHandshakeException

```
public class SSLHandshakeException extends SSLException {
// Public Constructors
    public SSLHandshakeException(String reason);
}
```

SSLKeyException**javax.net.ssl****Java 1.4*****serializable checked***

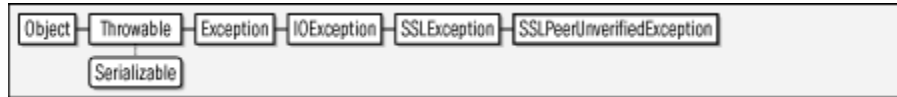
Signals a problem with the public key certificate and private key used by a server (or client) for authentication.

Figure 18-8. javax.net.ssl.SSLKeyException

```
public class SSLKeyException extends SSLException {
// Public Constructors
    public SSLKeyException(String reason);
}
```

SSLPeerUnverifiedException**javax.net.ssl****Java 1.4*****serializable checked***

Signals that authentication of the remote host was not successfully completed.

Figure 18-9. javax.net.ssl.SSLPeerUnverifiedException

```

public class SSLPeerUnverifiedException extends SSLException {
    // Public Constructors
    public SSLPeerUnverifiedException(String reason);
}

```

Thrown By

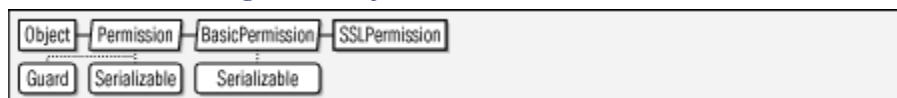
```

java.net.SecureCacheResponse.{getPeerPrincipal( ),
getServerCertificateChain( )},HandshakeCompletedEvent.
{getPeerCertificateChain( ),getPeerCertificates( ),
getPeerPrincipal( )},URLConnection.{getPeerPrincipal( ),
getServerCertificates( )},SSLSession.{getPeerCertificateChain( ),
getPeerCertificates( ),getPeerPrincipal( )}

```

SSLPermission**javax.net.ssl****Java 1.4*****serializable permission***

This Permission class controls access to sensitive methods in the javax.net.ssl package. The two defined target names are "setHostnameVerifier" and "getSSLSessionContext". The first is required in order to call `URLConnection.setHostnameVerifier()` and `URLConnection.setDefaultHostnameVerifier()`. The second permission target is required in order to call `SSLSession.getSessionContext()`.

Figure 18-10. javax.net.ssl.SSLPermission**Chapter 18. javax.net and javax.net.ssl**

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
 Print Publication Date: 2005/03/01

Prepared for Ronald Fischer, Safari ID: ronald.fischer@fussshuhn.de
 User number: 628024 Copyright 2008, Safari Books Online, LLC.

This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

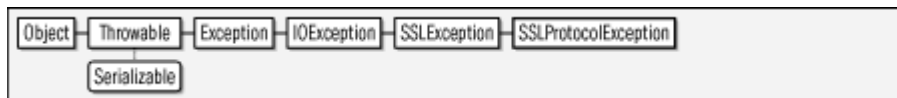
```

public final class SSLPermission extends java.security.BasicPermission {
    // Public Constructors
    public SSLPermission(String name);
    public SSLPermission(String name, String actions);
}

```

SSLProtocolException**javax.net.ssl****Java 1.4*****serializable checked***

Signals a problem at the SSL protocol level. An exception of this type usually indicates that there is a bug in the SSL implementation being used locally or on the remote host.

Figure 18-11. javax.net.ssl.SSLProtocolException

```

public class SSLProtocolException extends SSLException {
    // Public Constructors
    public SSLProtocolException(String reason);
}

```

SSLServerSocket**javax.net.ssl****Java 1.4**

This class is an SSL-enabled subclass of `java.net.ServerSocket` that is used to listen for and accept connections from clients and to create `SSLSocket` objects for communicating with those clients. Create an `SSLServerSocket` and bind it to a local port by calling one of the inherited `getServerSocket()` methods of an `SSLServerSocketFactory`. Once a `SSLServerSocket` is created, use it as you would a regular `ServerSocket`: call the inherited `accept()` method to wait for and accept a connection from a client, returning a `Socket` object. With `SSLServerSocket`, the `Socket` returned by `accept()` can always be cast to an instance of `SSLSocket`.

`SSLServerSocket` defines methods for setting the enabled protocols and cipher suites, and for querying the full set of supported protocols and suites. See `SSLSocket`, which has methods with the same names, for details. If your server desires or requires authentication

by its clients, call `setWantClientAuth()` or `setNeedClientAuth()`. These methods cause the `SSLSocket` objects returned by `accept()` to be configured to request or require client authentication.

In typical SSL networking scenarios, the client requires the server to provide authentication information. When you create an `SSLServerSocket` using the default `SSLServerSocketFactory`, the authentication information required is an X.509 public key certificate and the corresponding private key. The default `SSLServerSocketFactory` uses an `X509KeyManager` to obtain this information. The default `X509KeyManager` attempts to read this information from the `java.security.KeyStore` file specified by the system property `javax.net.ssl.keyStore`. It uses the value of the `javax.net.ssl.keyStorePassword` as the keystore password, and uses the value of the `javax.net.ssl.keyStoreType` system property to specify the keystore type. The key store should only contain valid keys and certificate chains that identify the server; the `X509KeyManager` automatically chooses a key and certificate chain that are appropriate for the client.

Figure 18-12. javax.net.ssl.SSLServerSocket



```

public abstract class SSLServerSocket extends java.net.ServerSocket {
    // Protected Constructors
    protected SSLServerSocket( ) throws java.io.IOException;
    protected SSLServerSocket(int port) throws java.io.IOException;
    protected SSLServerSocket(int port, int backlog) throws java.io.IOException;
    protected SSLServerSocket(int port, int backlog, java.net.InetAddress address)
        throws java.io.IOException;
    // Public Instance Methods
    public abstract String[ ] getEnabledCipherSuites( );
    public abstract String[ ] getEnabledProtocols( );
    public abstract boolean getEnableSessionCreation( );
    public abstract boolean getNeedClientAuth( );
    public abstract String[ ] getSupportedCipherSuites( );
    public abstract String[ ] getSupportedProtocols( );
    public abstract boolean getUseClientMode( );
    public abstract boolean getWantClientAuth( );
    public abstract void setEnabledCipherSuites(String[ ] suites);
    public abstract void setEnabledProtocols(String[ ] protocols);
    public abstract void setEnableSessionCreation(boolean flag);
    public abstract void setNeedClientAuth(boolean need);
    public abstract void setUseClientMode(boolean mode);
    public abstract void setWantClientAuth(boolean want);
}

```

SSLServerSocketFactory

javax.net.ssl

Java 1.4

This class is a `javax.net.ServerSocketFactory` for creating `SSLServerSocket` objects. Most applications use the default `SSLServerSocketFactory` returned by the static `getDefault()` method. Once this `SSLServerSocketFactory` has been obtained, they use one of the inherited `createServerSocket()` methods to create and optionally bind a new `SSLServerSocket`. The return value of the `createServerSocket()` methods is a `java.net.ServerSocket` object, but you can safely cast this object to a `SSLServerSocket` if you need to.

Applications that need to customize the SSL configuration and cannot use the default server socket factory may obtain a custom `SSLServerSocketFactory` from an `SSLContext`, which is essentially a factory for socket factories. See `SSLContext` for details.

Figure 18-13. `javax.net.ssl.SSLServerSocketFactory`



```

public abstract class SSLServerSocketFactory extends javax.net.ServerSocketFactory {
    // Protected Constructors
    protected SSLServerSocketFactory( );
    // Public Class Methods
    public static javax.net.ServerSocketFactory getDefault( );           synchronized
    // Public Instance Methods
    public abstract String[ ] getDefaultCipherSuites( );
    public abstract String[ ] getSupportedCipherSuites( );
}

```

Returned By

`SSLContext.getServerSocketFactory()`,
`SSLContextSpi.engineGetServerSocketFactory()`

SSLSession

javax.net.ssl

Java 1.4

A `SSLSession` object contains information about the SSL connection established through an `SSLSocket`. Use the `getSession()` method of a `SSLSocket` to obtain the `SSLSession` object for that socket. Many of the `SSLSession` methods return information that was obtained during the handshake phase of the connection. `getProtocol()` returns the specific version of the SSL or TLS protocol in use. `getCipherSuite()` returns the name of the cipher suite negotiated for the connection.

`getPeerHost()` returns the name of the remote host, and `getPeerCertificates()` returns the certificate chain, if any, that was received from the remote host during authentication. In Java 5.0 and later the peer's identity can also be queried with `getPeerPrincipal()`

The `invalidate()` method ends the session. It does not affect any current connections, but all future connections and any re-negotiations of existing connections will need to establish a new `SSLSession`. `isValid()` determines whether a session is still valid.

Multiple SSL connections between two hosts may share the same `SSLSession` as long as they are using the same protocol version and cipher suite. There is no way to enumerate the `SSLSocket` objects that share a session, but these sockets can exchange information by using `putValue()` to bind a shared object to some well-known name that can be looked up by other sockets with `getValue()`. `removeValue()` removes such a binding, and `getValueNames()` returns an array of all names that have objects bound to them in this session. Objects bound and unbound with `putValue()` and `removeValue()` may implement `SSLSessionBindingListener` to be notified when they are bound and unbound.

Note that the `getPeerCertificateChain()` method returns an object from the `javax.security.cert` package, which is not documented in this book. The method and package exist only for backward compatibility with earlier versions of the JSSE API, and should be considered deprecated. Use `getPeerCertificates()`, which uses `java.security.cert` instead.

```
public interface SSLSession {
    // Public Instance Methods
    5.0 int getApplicationBufferSize( );
        String getCipherSuite( );
        long getCreationTime( );
        byte[ ] getId( );
        long getLastAccessedTime( );
        java.security.cert.Certificate[ ] getLocalCertificates( );
    5.0 java.security.Principal getLocalPrincipal( );
    5.0 int getPacketBufferSize( );
        javax.security.cert.X509Certificate[ ] getPeerCertificateChain( )
            throws SSLPeerUnverifiedException;
        java.security.cert.Certificate[ ] getPeerCertificates( )
            throws SSLPeerUnverifiedException;
        String getPeerHost( );
    5.0 int getPeerPort( );
    5.0 java.security.Principal getPeerPrincipal( ) throws SSLPeerUnverifiedException;
        String getProtocol( );
        SSLSessionContext getSessionContext( );
        Object getValue(String name);
        String[ ] getValueNames( );
        void invalidate( );
    5.0 boolean isValid( );
        void putValue(String name, Object value);
        void removeValue(String name);
}
```


Passed To

```
HandshakeCompletedEvent.HandshakeCompletedEvent( ),
HostnameVerifier.verify( ),
SSLSessionBindingEvent.SSLSessionBindingEvent( )
```

Returned By

```
HandshakeCompletedEvent.getSession( ), SSLEngine.getSession( ),
SSLSessionBindingEvent.getSession( ),
SSLSessionContext.getSession( ), SSLSocket.getSession( )
```

SSLSessionBindingEvent**javax.net.ssl****Java 1.4*****serializable event***

An object of this type is passed to the `valueBound()` and `valueUnbound()` methods of an object that implements `SSLSessionBindingListener` when that object is bound or unbound in a `SSLSession` with the `putValue()` or `removeValue()` methods of `SSLSession`. `getName()` returns the name to which the object was bound or unbound, and `getSession()` returns the `SSLSession` object in which the binding was created or removed.

Figure 18-14. javax.net.ssl.SSLSessionBindingEvent

```
public class SSLSessionBindingEvent extends java.util.EventObject {
// Public Constructors
    public SSLSessionBindingEvent(SSLSession session, String name);
// Public Instance Methods
    public String getName( );
    public SSLSession getSession( );
}
```

Passed To

```
SSLSessionBindingListener.{valueBound( ), valueUnbound( )}
```

SSLSessionBindingListener**javax.net.ssl****Java 1.4*****event listener***

This interface is implemented by an object that wants to be notified when it is bound or unbound in an `SSLSession` object. If the object passed to the `putValue()` method of

a `SSLSession` implements this interface, then its `valueBound()` method will be called by `putValue()`, and its `valueUnbound()` method will be called when that object is removed from the `SSLSession` with `removeValue()` or when it is replaced with a new object by `putValue()`. The argument to both methods of this interface is a `SSLSessionBindingEvent`, which specifies both the name to which the object was bound or unbound, and the `SSLSession` within which it was bound or unbound.

Figure 18-15. javax.net.ssl.SSLSessionBindingListener



```

public interface SSLSessionBindingListener extends java.util.EventListener {
    // Public Instance Methods
    void valueBound(SSLSessionBindingEvent event);
    void valueUnbound(SSLSessionBindingEvent event);
}

```

SSLSessionContext

javax.net.ssl

Java 1.4

A `SSLSessionContext` groups and controls `SSLSession` objects. It is a low-level interface and is not commonly used in application code. `getIds()` returns an Enumeration of session IDs, and `getSession()` returns the `SSLSession` object associated with one of those IDs. `setSessionCacheSize()` specifies the total number of concurrent sessions allowed in the group, and `setSessionTimeout()` specifies the timeout length for those sessions. An `SSLSessionContext` can serve as a cache for `SSLSession` objects, facilitating reuse of those objects for multiple connections between the same two hosts.

Providers are not required to support this interface. Those that do return an implementing object from the `getSessionContext()` method of an `SSLSession` object, and also return implementing objects from the `getClientSessionContext()` and `getServerSessionContext()` methods of an `SSLContext` object, providing separate control over client and server SSL connections.

```

public interface SSLSessionContext {
    // Public Instance Methods
    java.util.Enumeration getIds();
    SSLSession getSession(byte[] sessionId);
    int getSessionCacheSize();
    int getSessionTimeout();
    void setSessionCacheSize(int size) throws IllegalArgumentException;
    void setSessionTimeout(int seconds) throws IllegalArgumentException;
}

```

Chapter 18. javax.net and javax.net.ssl

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Prepared for Ronald Fischer, Safari ID: ronald.fischer@fussshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.

This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

Returned By

```
SSLContext.{getClientSessionContext( ),
getServerSessionContext( )}, SSLContextSpi.
{engineGetClientSessionContext( ),
engineGetServerSessionContext( )}, SSLSession.getSessionContext( )
```

SSLSocket**javax.net.ssl****Java 1.4**

An `SSLSocket` is a "secure socket" subclass of `java.net.Socket` that implements the SSL or TLS protocols, which are commonly used to authenticate a server to a client and to encrypt the data transferred between the two. Create a `SSLSocket` for connecting to a SSL-enabled server by calling one of the `createSocket()` methods of a `SSLSocketFactory` object. See `SSLSocketFactory` for details. If you are writing server code, then you will obtain a `SSLSocket` for communicating with an SSL-enabled client from the inherited `accept()` method of an `SSLServerSocket`. See `SSLServerSocket` for details.

`SSLSocket` inherits all of the standard socket method of its superclass, and can be used for networking just like an ordinary `java.net.Socket` object. In addition, however, it also defines methods that control how the secure connection is established. These methods may be called before the SSL "handshake" occurs. The handshake does not occur when the socket is first created and connected, so that you can configure various SSL parameters that control how the handshake occurs. Calling `startHandshake()`, `getSession()`, or reading or writing data on the socket trigger a handshake, so you must configure the socket before doing any of these things. If you want to be notified when the handshake occurs, call `addHandshakeCompletedListener()` to register a listener object to receive the notification.

`getSupportedProtocols()` returns a list of secure socket protocols that are supported by the socket implementation. `setEnabledProtocols()` allows you to specify the name or names of the supported protocols that you are willing to use for this socket. `getSupportedCipherSuite()` returns the full set of cipher suites supported by the underlying security provider. `setEnabledCipherSuites()` specifies a list of one or more cipher suites that you are willing to use for the connection. Note that not all supported cipher suites are enabled by default: only suites that provide encryption and require the server to authenticate itself to the client are enabled. If you want to allow the

server to remain anonymous, you can use `setEnabledCipherSuites()` to enable a nonauthenticating suite. Specific protocols and cipher suites are not described here because using them correctly requires a detailed understanding of cryptography, which is beyond the scope of this reference. Most applications can simply rely on the default set of enabled protocols and cipher suites.

If you are writing a server and have obtained an `SSLSocket` by accepting a connection on an `SSLServerSocket`, then you may call `setWantClientAuth()` to request that the client authenticate itself to you, and you may call `setNeedClientAuth()` to require that the client authenticate itself during the handshake. Note, however, that it is usually more efficient to request or require client authentication on the server socket than it is to call these methods on each `SSLSocket` it creates.

The configuration methods described above must be called before the SSL handshake occurs. Call `getSession()` to obtain an `SSLSession` object that you can query for information about the handshake, such as the protocol and cipher suite in use, and the identity of the server. Note that a call to `getSession()` will cause the handshake to occur if it has not already occurred, so you can call this method at any time.

Figure 18-16. javax.net.ssl.SSLSocket



```

public abstract class SSLSocket extends java.net.Socket {
// Protected Constructors
    protected SSLSocket( );
    protected SSLSocket(String host, int port)
        throws java.io.IOException, java.net.UnknownHostException;
    protected SSLSocket(java.net.InetAddress address, int port)
        throws java.io.IOException;
    protected SSLSocket(String host, int port, java.net.InetAddress clientAddress,
        int clientPort) throws java.io.IOException,
        java.net.UnknownHostException;
    protected SSLSocket(java.net.InetAddress address, int port, java.net.InetAddress clientAddress,
        int clientPort) throws java.io.IOException;
// Event Registration Methods (by event name)
    public abstract void addHandshakeCompletedListener(HandshakeCompletedListener listener);
    public abstract void removeHandshakeCompletedListener(HandshakeCompletedListener listener);
// Public Instance Methods
    public abstract String[ ] getEnabledCipherSuites( );
    public abstract String[ ] getEnabledProtocols( );
    public abstract boolean getEnableSessionCreation( );
    public abstract boolean getNeedClientAuth( );
    public abstract SSLSession getSession( );
    public abstract String[ ] getSupportedCipherSuites( );
    public abstract String[ ] getSupportedProtocols( );
    public abstract boolean getUseClientMode( );
    public abstract boolean getWantClientAuth( );
    public abstract void setEnabledCipherSuites(String[ ] suites);
    public abstract void setEnabledProtocols(String[ ] protocols);
    public abstract void setEnableSessionCreation(boolean flag);
    public abstract void setNeedClientAuth(boolean need);
    public abstract void setUseClientMode(boolean mode);
    public abstract void setWantClientAuth(boolean want);
    public abstract void startHandshake( ) throws java.io.IOException;
}
  
```

Passed To

HandshakeCompletedEvent.HandshakeCompletedEvent ()

Returned By

HandshakeCompletedEvent.getSocket ()

SSLSocketFactory**javax.net.ssl****Java 1.4**

This class is a `javax.net.SocketFactory` for creating `SSLSocket` objects. Most applications use the default `SSLSocketFactory` returned by the static `getDefault ()` method. Once this `SSLSocketFactory` has been obtained, they use one of the inherited `createSocket ()` methods to create, and optionally connect and bind, a new `SSLSocket`. The return value of the `createSocket ()` methods is a `java.net.Socket` object, but you can safely cast this object to a `SSLSocket` if you need to. `SSLSocketFactory` defines one new version of `createSocket ()` in addition to the ones it inherits from its superclass. This version of the method creates an `SSLSocket` that is layered over an existing `Socket` object rather than creating a new socket entirely from scratch.

Applications that need to customize the SSL configuration and cannot use the default socket factory may obtain a custom `SSLSocketFactory` from an `SSLContext`, which is essentially a factory for socket factories. See `SSLContext` for details.

Figure 18-17. javax.net.ssl.SSLSocketFactory



```

public abstract class SSLSocketFactory extends javax.net.SocketFactory {
// Public Constructors
    public SSLSocketFactory ( );
// Public Class Methods
    public static javax.net.SocketFactory getDefault ( );           synchronized
// Public Instance Methods
    public abstract java.net.Socket createSocket(java.net.Socket s, String host,
        int port, boolean autoClose)
        throws java.io.IOException;
    public abstract String[] getDefaultCipherSuites ( );
    public abstract String[] getSupportedCipherSuites ( );
}

```

Passed To

HttpsURLConnection.{setDefaultSSLSocketFactory (),
setSSLSocketFactory () }

Returned By

```
HttpsURLConnection.{getDefaultSSLSocketFactory( ),
getSSLSocketFactory( )}, SSLContext.getSocketFactory( ),
SSLContextSpi.engineGetSocketFactory( )
```

TrustManager**javax.net.ssl****Java 1.4**

This is a marker interface to identify trust manager objects. A trust manager is responsible for examining the authentication credentials (such as a certificate chain) presented by the remote host and deciding whether to trust those credentials and accept them. A `TrustManager` is usually used an SSL client to decide whether the SSL server is authentic, but may also be used by an SSL server when client authentication is also required.

Use a `TrustManagerFactory` to obtain `TrustManager` objects. `TrustManager` objects returned by a `TrustManagerFactory` can always be cast to a subinterface specific to a specific type of keys. See `X509TrustManager`, for exmaple.

```
public interface TrustManager {
}
```

Implementations

`X509TrustManager`

Passed To

```
SSLContext.init( ), SSLContextSpi.engineInit( )
```

Returned By

```
TrustManagerFactory.getTrustManagers( ),
TrustManagerFactorySpi.engineGetTrustManagers( )
```

TrustManagerFactory**javax.net.ssl****Java 1.4**

A `TrustManagerFactory` is responsible for creating `TrustManager` objects for a specific trust management algorithm. Obtain a `TrustManagerFactory` object by calling one of the `getInstance()` methods and specifying the desired algorithm and, optionally, the desired provider. In Java 1.4, the "SunX509" algorithm is the only one supported by the default "SunJSSE" provider. After calling `getInstance()`, you initialize the factory object with `init()`. For the "SunX509" algorithm, you pass a

KeyStore object to `init()`. This KeyStore should contain the public keys of trusted CAs (certification authorities). Once a TrustManagerFactory has been created and initialized, use it to create a TrustManager by calling `getTrustManagers()`. This method returns an array of TrustManager objects because some trust management algorithms may handle more than one type of key or certificate. The "SunX509" algorithm manages only X.509 keys, and always returns an array with an X509TrustManager object as its single element. This returned array is typically passed to the `init()` method of an SSLContext object.

If no KeyStore is passed to the `init()` method of the TrustManagerFactory for the "SunX509" algorithm, then the factory uses a KeyStore created from the file named by the system property `javax.net.ssl.trustStore` if that property is defined. (It also uses the key store type and password specified by the properties `javax.net.ssl.trustStoreType` and `javax.net.ssl.trustStorePassword`.) Otherwise, it uses the file `jre/lib/security/jssecacerts` in the Java distribution, if it exists. Otherwise it uses the file `jre/lib/security/cacerts` which is part of Sun's Java distribution. Sun ships a default `cacerts` file that contains certificates for several well-known and reputable CAs. You can use the `keytool` program to edit the `cacerts` keystore (the default password is "changeit").

```
public class TrustManagerFactory {
    // Protected Constructors
    protected TrustManagerFactory(TrustManagerFactorySpi factorySpi, java.security.
        Provider provider, String algorithm);

    // Public Class Methods
    public static final String getDefaultAlgorithm();
    public static final TrustManagerFactory getInstance(String algorithm)
        throws java.security.NoSuchAlgorithmException;
    public static final TrustManagerFactory getInstance(String algorithm,
        java.security.Provider provider)
        throws java.security.NoSuchAlgorithmException;
    public static final TrustManagerFactory getInstance(String algorithm,
        String provider) throws java.security.NoSuchAlgorithmException,
        java.security.NoSuchProviderException;

    // Public Instance Methods
    public final String getAlgorithm();
    public final java.security.Provider getProvider();
    public final TrustManager[] getTrustManagers();
    public final void init(ManagerFactoryParameters spec)
        throws java.security.InvalidAlgorithmParameterException;
    public final void init(java.security.KeyStore ks)
        throws java.security.KeyStoreException;
}
```

TrustManagerFactorySpi

javax.net.ssl

Java 1.4

This abstract class defines the Service Provider Interface for `TrustManagerFactory`. Security providers must implement this interface, but applications never need to use it.

```
public abstract class TrustManagerFactorySpi {
    // Public Constructors
    public TrustManagerFactorySpi( );
    // Protected Instance Methods
    protected abstract TrustManager[ ] engineGetTrustManagers( );
    protected abstract void engineInit(ManagerFactoryParameters spec)
        throws java.security.InvalidAlgorithmParameterException;
    protected abstract void engineInit(java.security.KeyStore ks)
        throws java.security.KeyStoreException;
}
```

Passed To

`TrustManagerFactory.TrustManagerFactory()`

X509ExtendedKeyManager

javax.net.ssl

Java 5.0

This class implements the `X509KeyManager` interface and extends it with two methods.

Figure 18-18. javax.net.ssl.X509ExtendedKeyManager



```
public abstract class X509ExtendedKeyManager implements X509KeyManager {
    // Protected Constructors
    protected X509ExtendedKeyManager( );
    // Public Instance Methods
    public String chooseEngineClientAlias(String[ ] keyType,
        java.security.Principal[ ] issuers,
        SSLEngine engine);    constant
    public String chooseEngineServerAlias(String keyType,
        java.security.Principal[ ] issuers,
        SSLEngine engine);    constant
}
```

X509KeyManager

javax.net.ssl

Java 1.4

This interface is a `KeyManager` for working with X.509 certificates. An `X509KeyManager` is used during the SSL handshake by a peer that authenticates itself by

providing an X.509 certificate chain to the remote host. This is usually done on the server side of the SSL connection, and can be done on the client-side as well, although that is uncommon. Obtain an `X509KeyManager` object either by implementing your own or from a `KeyManagerFactory` created with an algorithm of "SunX509". Applications do not call the methods of an `X509KeyManager` themselves. Instead, they simply supply an appropriate `X509KeyManager` object to the `SSLContext` object that is responsible for setting up SSL connections. When the system needs to authenticate itself during an SSL handshake, it calls various methods of the key manager object to obtain the information in needs.

An `X509KeyManager` retrieves keys and certificate chains from the `KeyStore` object that was passed to the `init()` method of the `KeyManagerFactory` object from which it was created. `getPrivateKey()` and `getCertificateChain()` return the private key and the certificate chain for a specified alias. The other methods are called to list all aliases in the keystore or to choose one alias from the keystore that matches the specified keytype and certificate authority criteria. In this way, a `X509KeyManager` can choose a certificate chain (and its corresponding key) based on the types of keys and the list of certificate authorities recognized by the remote host.

Figure 18-19. javax.net.ssl.X509KeyManager



```
public interface X509KeyManager extends KeyManager {
    // Public Instance Methods
    String chooseClientAlias(String[] keyType, java.security.Principal[] issuers,
        java.net.Socket socket);
    String chooseServerAlias(String keyType, java.security.Principal[] issuers,
        java.net.Socket socket);
    java.security.cert.X509Certificate[] getCertificateChain(String alias);
    String[] getClientAliases(String keyType, java.security.Principal[] issuers);
    java.security.PrivateKey getPrivateKey(String alias);
    String[] getServerAliases(String keyType, java.security.Principal[] issuers);
}
```

Implementations

`X509ExtendedKeyManager`

X509TrustManager

javax.net.ssl

Java 1.4

This interface is a `TrustManager` for working with X.509 certificates. Trust managers are used during the handshake phase of SSL connection to determine whether the authentication credentials presented by the remote host are trusted. This is usually done

on the client-side of an SSL connection, but may also be done on the server side. Obtain an `X509TrustManager` either by implementing your own or from a `TrustManagerFactory` that was created to use the "SunX509" algorithm. Applications do call the methods of this interface themselves; instead, they simply provide an appropriate `X509TrustManager` object to the `SSLContext` object that is responsible for setting up SSL connections. When the system needs to determine whether the authentication credentials presented by the remote host are trusted, it calls the methods of the trust manager.

Figure 18-20. javax.net.ssl.X509TrustManager



```

public interface X509TrustManager extends TrustManager {
    // Public Instance Methods
    void checkClientTrusted(java.security.cert.X509Certificate[ ] chain,
        String authType) throws java.security.cert.CertificateException;
    void checkServerTrusted(java.security.cert.X509Certificate[ ] chain,
        String authType) throws java.security.cert.CertificateException;
    java.security.cert.X509Certificate[ ] getAcceptedIssuers( );
}

```