Avid.

Using Netbeans 5.5 and Sun Metro with Avid Interplay WS

Revision 1.3 – December 5, 2007

Overview

The Interplay WS API provides SOAP web services for interacting with Avid Interplay. The first release provides operations for exchanging and modifying metadata in the Avid Interplay Engine. This document demonstrates how to create a Java client for Interplay WS using the NetBeans 5.5 and Sun's Metro JAX-WS implementation.

The Interplay WS distribution also comes with a *screencast* demonstrating how to use NetBeans with Interplay WS. You may want to watch the screencast for a more visual and dynamic demonstration.

Sun Metro and Java SOAP Toolkits

Sun's Metro project was chosen for this tutorial due to its ease of use (via NetBeans integration) and excellent interoperability. There are also many other Java SOAP toolkits available, including: Axis, Axis2, CXF, JBossWS, and XFire.

General Conventions Used in this Document

Server name and port
The server name and port of the Interplay WS services will be different for each site. For the purposes of this documentation, we will be using the *localhost* server and port *80* (the default port). You should modify your work to reflect the actual hostname and port of your Interplay WS installation.

For simplicity, this documentation will mainly use *http*. Instructions for using *https* can be found near the end of this document. For security purposes, we recommend you use *https* in production environments.

Workgroup name and server mapping
The Interplay WS application must be configured with workgroup name and server mappings. This can be done via the Avid Interplay Framework Configuration client. For the purposes of this documentation, we will assume that the workgroup name *WGA* is mapped to our Interplay Engine server.

Locating the WSDL and XSD files

Since Interplay WS is a SOAP web service, it uses a Web Services Description Language (WSDL) file to describe its interface. This WSDL also references an XML Schema Document (XSD) to define the XML types used in the messages. In development, your SOAP toolkit will use the WSDL file to generate client code to access the service.

The WSDL for Interplay WS Assets can be found at:
http://iws-srv/services/Assets?wsdl
(Remember to change the server name and port to match your environment)

This WSDL will also reference the XSD, which can be found at:
http://iws-srv/services/Assets?xsd=assets.xsd

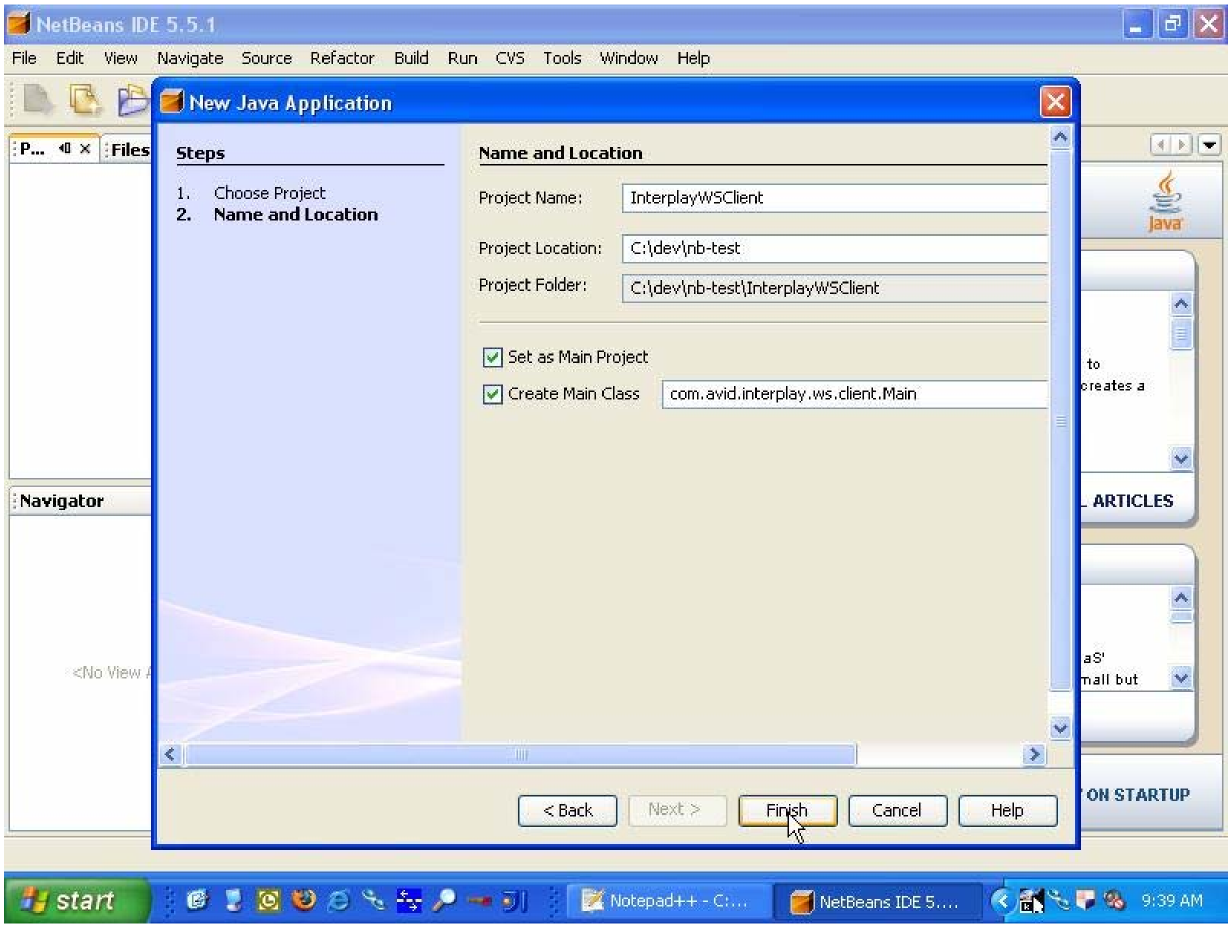
Creating a Metro Client with NetBeans Step-by-Step

For this tutorial, we will create a simple Java Console application that gets the children of the *Projects/Rainforest* folder.

Create a New Java Application Project

1. Create a new project by choosing *File -> New Project…*

2. Choose *General* and *Java Application*, and then enter the name, location, and package of the project.



Project Name: InterplayWSClient

Project Location: C:\dev\nb-test

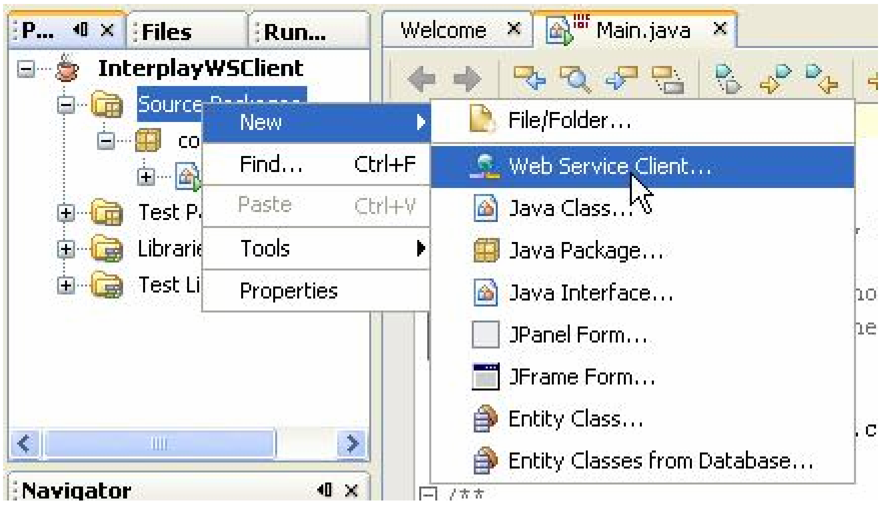
Project Folder: C:\dev\nb-test\InterplayWSClient

Set as Main Project ✓

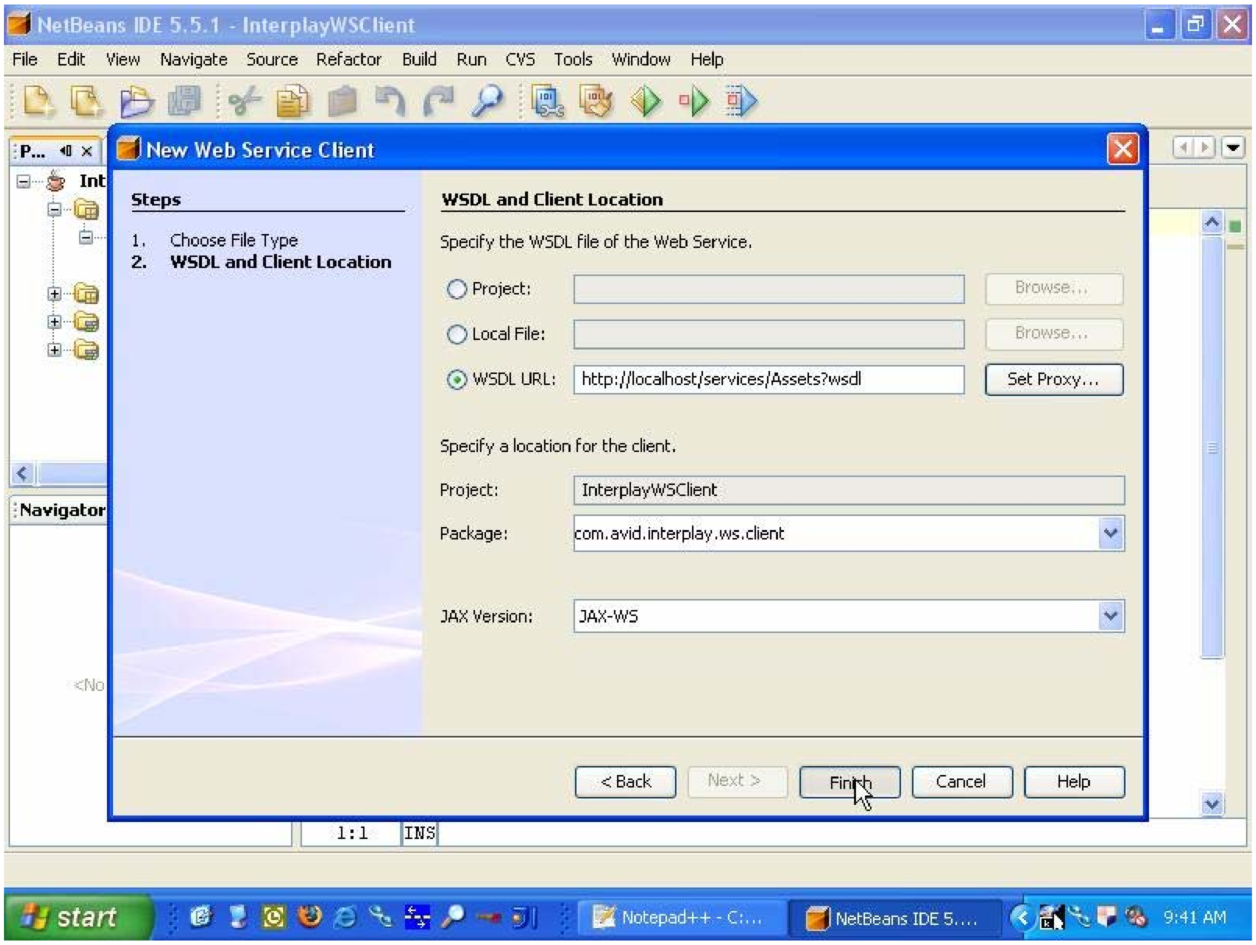
Create Main Class ✓ com.avid.interplay.ws.client.Main

Create a New Web Service Client for the Interplay WS Assets Service

1. Right-click in the *Projects* pane and choose *New -> Web Service Client…*



2. Type in the URL to the WSDL file (see *Locating the WSDL and XSD File* above). Then choose the package, make sure that *JAX-WS* is selected, and click *Finish.*


NetBeans IDE 5.5.1 - InterplayWSClient
File Edit View Navigate Source Refactor Build Run CVS Tools Window Help
New Web Service Client
Steps
1. Choose File Type
2. WSDL and Client Location
< Back
Next >
Finish
Cancel
Help
start
Notepad++ - C:...
NetBeans IDE 5....
9:41 AM

WSDL and Client Location

Specify the WSDL file of the Web Service.

Project:

Local File:

WSDL URL: http://localhost/services/Assets?wsdl

Specify a location for the client.

Project: InterplayWSClient

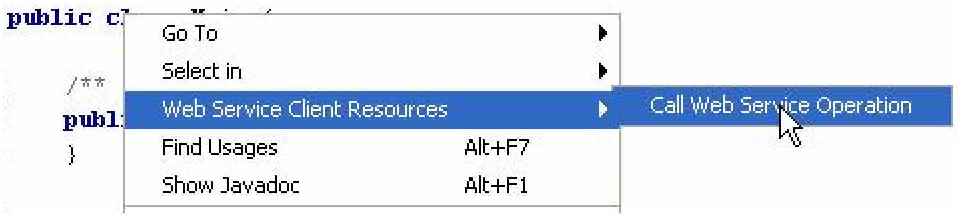
Package: com.avid.interplay.ws.client

JAX Version: JAX-WS

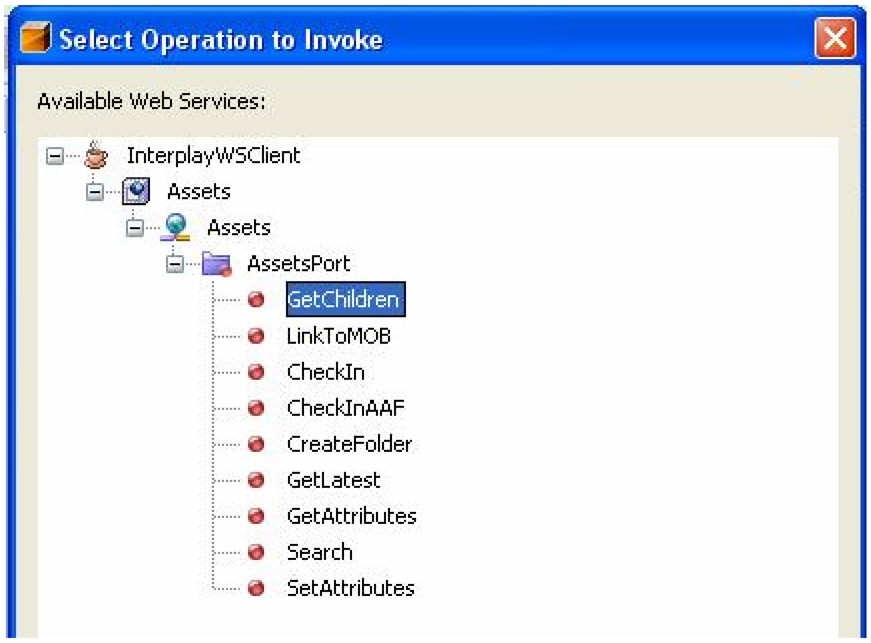
Generating Starting Point Code for Calling an Operation

NetBeans has a wizard for generating starting point code to call an operation. Although implementation is easy enough without the wizard, we'll show you how to use it. If you want to just write the code from scratch, skip to the next section.

1. Right-click in the *main* method and choose *Web Service Client Resources* -> *Call Web Service Operation*.



2. Choose the *GetChildren* operation and then click OK.



3. Now fill in the remaining code where the comments are marked *TODO*. You may also want to make the code look cleaner by changing fully qualified names to use import statements instead.

4. For more details on filling in the rest of the code, please refer to the next section: *Write Code to Use the Interplay WS Assets Service*.

Write Code to Use the Interplay WS Assets Service

1. Get an implementation of the *Assets* port. For this example, we are writing this and all other code in the *main* method of our program.

```
Assets service = new Assets();
AssetsPortType port = service.getAssetsPort();
```

2. Create and populate the parameters for the *GetChildren* operation. See the Interplay WS Reference Guide for more information on Interplay URIs.

```
GetChildrenType body = new GetChildrenType();
body.setInterplayURI("interplay://WGA/Projects/Rainforest");
body.setIncludeFiles(true); // booleans default to false if not specified
body.setIncludeFolders(true);
body.setIncludeMOBs(true);
```

3. Set the *UserCredentials* to pass in the operation.

```
UserCredentialsType creds = new UserCredentialsType();
creds.setUsername("jsmith");
creds.setPassword("secret");
```

4. Execute the *GetChildren* operation via the *Assets* port.

```
GetChildrenResponseType result = port.getChildren(body, creds);
```

5. Handle the results. For this example, we'll just output the Interplay URI and the returned attributes. *See the Interplay WS Reference Guide for more info on attributes.*

```
for (AssetDescriptionType asset : result.getResults().getAssetDescription()) {
    System.out.println(asset.getInterplayURI());
    for (AttributeType att : asset.getAttributes().getAttribute()) {
        System.out.println(att.getGroup() + "." + att.getName() + " = " +
                           att.getValue());
    }
    System.out.println();
}
```

6. Save and run the program. You should see the results in the *Output* pane at the bottom of the NetBeans window.

Advanced Topics

Enabling MTOM Support

MTOM is the recommended best practice for efficiently transporting binary files via SOAP. Enabling MTOM in Metro is very easy.

1. Add the import statement for MTOMFeature at the top of your Java class.

```
import javax.xml.ws.soap.MTOMFeature;
```

2. In your custom code, you must now pass in the MTOMFeature when getting the *Assets* port.

```
AssetsPortType port = service.getAssetsPort(new MTOMFeature());
```

Changing the Endpoint URL

In some cases, you may want to change the endpoint that your client code uses. By default, the code will use whichever endpoint URL was specified in the WSDL to create the service. To make your client communicate with an Interplay WS service on a different host, you need to change the endpoint URL.

Unfortunately, Metro does not provide a simple way to change the endpoint URL. It is possible, but the statement is a bit more complicated than it seems it should be.

```
((javax.xml.ws.BindingProvider)port).getRequestContext().put(
        javax.xml.ws.BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
        "http://iws-srv2/services/Assets");
```

Another common reason to change the endpoint is for debugging purposes. You can use a tool such as TCPMon to create a relay to the Interplay WS host. You would then configure your client code to send requests to TCPMon on your local machine. TCPMon could then intercept the sent and received messages so you can view their contents. If TCPMon is listening on port 81 on your local machine, your code should look like this:

```
((javax.xml.ws.BindingProvider)port).getRequestContext().put(
        javax.xml.ws.BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
        "http://localhost:81/services/Assets");
```

Enabling Secure Transmission over HTTPS

The Interplay WS service also allows for encrypted transmissions via HTTPS. The Interplay WS server uses a *self-signed certificate*, so Java will not trust it by default. There are two ways to get Java to trust the Interplay WS service.

Importing the Interplay WS Certificate into your KeyStore

The recommended approach is to use Java's keytool to import the Interplay WS server's certificate into your keystore. This is the more complicated of the two approaches, but is considered more secure. For your convenience, there is an example Java utility at the end of this documentation that shows how to download the certificate or programmatically add it to the client keystore.

Configuring Java SSL to Trust Self-Signed Certificates

You can also configure Java to override the default TrustManager. This is a system wide setting, so it may affect the security of other applications running within the JVM.

1. Create a custom *TrustManager* that trusts the self-signed certificate.

```
javax.net.ssl.TrustManager[] interplayWsTrustingManager = new javax.net.ssl.TrustManager[]{
    new javax.net.ssl.X509TrustManager() {
        public java.security.cert.X509Certificate[] getAcceptedIssuers() { return null; }
        public void checkClientTrusted(java.security.cert.X509Certificate[] certs, String authType) { }
        public void checkServerTrusted(java.security.cert.X509Certificate[] certs, String authType)
                throws java.security.cert.CertificateException {
            // Do certificate validation logic here.
            // This example trusts any certs that have an Organization Unit of "Interplay"
            for (java.security.cert.X509Certificate cert : certs) {
                if (cert.getSubjectX500Principal().getName().contains("OU=Interplay")) return;
            }
            // No certs passed the test, so throw a CertificateException to indicate NO TRUST
            throw new java.security.cert.CertificateException();
        }
    }
};
```

2. Configure the SSLContext and HttpsUrlConnection classes.

```
try{
    SSLContext sc = SSLContext.getInstance("SSL");
    sc.init(null, interplayWsTrustingManager, new java.security.SecureRandom());
    HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
} catch (Exception e) {
    e.printStackTrace();
}
```

3. Change your *Assets* port to use HTTPS instead of HTTP.

```
port.Url = "https://iws-srv/services/Assets";
```

Changing the Timeout Length for SOAP Requests

Some queries (particularly broad search queries) may take some time before they return with a response. You can configure your client to timeout after a certain length of time by adding a value to the request context. Timeout values are measured in milliseconds.

```
((BindingProvider)port).getRequestContext().put("com.sun.xml.ws.request.timeout", 5*60*1000);
```

This essentially calls setReadTimeout(int …) on the java.net.URLConnection class. If a timeout does occur in your client, a java.net.SocketTimeoutException will be thrown.

If you do not wish for timeouts to occur, you can disable timeouts by setting the timeout value to 0:

```
((BindingProvider)port).getRequestContext().put("com.sun.xml.ws.request.timeout", 0);
```

Logging Outgoing and Incoming Message Traffic

Sometimes it is helpful to see the SOAP XML messages for debugging purposes. This can be done quite easy using Sun Metro. Once configured, the incoming and outgoing messages will be logged to standard out.

To log messages using Sun Metro:

1. Set the HttpTransportPipe to dump its messages:

```
com.sun.xml.ws.transport.http.client.HttpTransportPipe.dump=true;
```

CertificateUtility.java

The following is an example Java utility for extracting a certificate from an HTTPS server and downloading it to a file or importing it to a keystore. This code is provided *as an example only* and is not intended for production use. We have included it since it may be helpful in your development.

```
/*
 *  This is an example of how to programmatically get an X509 certificate from a running
 *  HTTPS server and download it as a file or import it into a keystore.  This example comes
 *  with no warranty expressed or implied.
 *
 *  To use its interactive commandline, compile and run this class.  Please review the code
 *  to fully understand how it works.
 */
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;
import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.Certificate;
import java.security.cert.CertificateEncodingException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.Arrays;
import java.util.Enumeration;

public class CertificateUtility {
    BufferedReader in;

    public CertificateUtility() {
        in = new BufferedReader(new InputStreamReader(System.in));
    }

    Certificate[] getServerCertificates(String host, int port) {

        Certificate[] certs = new Certificate[0];

        // Create a trust manager that does not validate certificate chains
        TrustManager[] allTrustingTrustManagers = new TrustManager[]{
            new X509TrustManager() {
                public java.security.cert.X509Certificate[] getAcceptedIssuers() { return null; }
                public void checkClientTrusted(X509Certificate[] certs, String authType) { }
                public void checkServerTrusted(X509Certificate[] certs, String authType) { }
            }
        };

        SSLSocket socket = null;
        try {
            // Get an all trusting socket factory
            SSLContext sslContext = SSLContext.getInstance("SSL");
            sslContext.init(null, allTrustingTrustManagers, null);
            SSLSocketFactory factory = sslContext.getSocketFactory();
            socket = (SSLSocket)factory.createSocket(host, port);

            // Connect to the server and get the certificate chain
            socket.startHandshake();
            certs = socket.getSession().getPeerCertificates();
        } catch (Exception e) {
            System.out.println("There was an error connecting to the host.  Reason: " + e.getMessage());
        } finally {
            // Close the socket
            try { if (socket != null) socket.close(); } catch (IOException e) { }
        }

        return certs;
    }

    KeyStore getKeyStore(File ksFile, String ksPassword)
    throws KeyStoreException, NoSuchAlgorithmException, IOException, CertificateException {

        KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());
        FileInputStream fis = null;
        try {
            fis = ksFile.exists() ? new FileInputStream(ksFile) : null;
            ks.load(fis, ksPassword.toCharArray());
        } catch (FileNotFoundException e) {
            // This should never happen since we check if it exists first
        } finally {
            try { if (fis != null) { fis.close(); } } catch (IOException e) {}
        }

        return ks;
    }

    private void importCertInKeyStore(Certificate cert, KeyStore ks, String alias)
    throws KeyStoreException {
        ks.setCertificateEntry(alias, cert);
    }

    private void saveKeyStore(KeyStore ks, File ksFile, String password)
    throws NoSuchAlgorithmException, IOException, CertificateException, KeyStoreException {
        FileOutputStream fos = new FileOutputStream(ksFile);
        ks.store(fos, password.toCharArray());
    }

    public void interactiveSession() throws IOException {
        String host = promptForInput("Hostname: ");
        int port = Integer.valueOf(promptForInput("Port: "));

        Certificate[] certs = getServerCertificates(host, port);
        if (certs.length == 0) {
            System.out.println("No server certificates were found for " + host + " on port " + port);
            return;
        }

        String option =
                promptForInput("Would you like to [d]ownload or [i]nstall the certificate?").trim();
        if ("i".equalsIgnoreCase(option)) {
            interactiveImport(host, certs);
        } else {
            interactiveDownload(certs);
        }
    }

    public void interactiveDownload(Certificate[] certs) throws IOException {
        try {
            for (Certificate cert: certs) {
                if (cert instanceof X509Certificate) {
                    byte[] encoded = cert.getEncoded();
                    ByteArrayInputStream bais = new ByteArrayInputStream(encoded);
                    String name = "cert" + System.currentTimeMillis() + ".cer";
                    FileOutputStream fos = new FileOutputStream(name);
                    byte[] buf = new byte[1024];
                    int i;
                    while((i=bais.read(buf))!=-1) {
                        fos.write(buf, 0, i);
                    }
                    bais.close();
                    fos.close();
                    printCertificateDetails((X509Certificate) cert);
                    System.out.println("Successfully saved certificate to " + name);
                    return;
                }
            }
        } catch (CertificateEncodingException e) { }
```

```
            // Must have been a problem
            System.out.println("There was an error downloading the certificate.");
        }

        public void interactiveImport(String host, Certificate[] certs) throws IOException {
            KeyStore ks = null;
            File ksFile = null;
            String ksPassword = null;
            while (ks == null) {
                while (ksFile == null) {
                    String type = promptForInput("Import to [j]ssecacerts or [c]ustom truststore? ").trim();
                    if ("j".equalsIgnoreCase(type)) {
                        File cacerts = new File(System.getProperty("java.home") +
                                "/jre/lib/security/cacerts");
                        if (! cacerts.exists()) {
                            cacerts = new File(System.getProperty("java.home") + "/lib/security/cacerts");
                        }
                        if (cacerts.exists()) {
                            File jssecacerts = new File(cacerts.getAbsolutePath().replace("cacerts",
                                    "jssecacerts"));
                            if (! jssecacerts.exists()) {
                                // Copy cacerts to jssecacerts
                                System.out.println("Copying " + cacerts.getAbsolutePath() +
                                        " to jssecacerts...");
                                FileInputStream fis  = new FileInputStream(cacerts);
                                FileOutputStream fos = new FileOutputStream(jssecacerts);
                                byte[] buf = new byte[1024];
                                int i;
                                while((i=fis.read(buf))!=-1) {
                                    fos.write(buf, 0, i);
                                }
                                fis.close();
                                fos.close();
                            }
                            ksFile = jssecacerts;
                        }
                    }
                    if (ksFile == null) {
                        String ksPath =
                                promptForInput("Path to keystore file [<enter> for ~/.keystore]: ").trim();
                        if (ksPath.length() == 0) {
                            ksPath = System.getProperty("user.home") + "/.keystore";
                        }
                        ksFile = new File(ksPath);
                        if (! ksFile.exists()) {
                            if (! promptForBoolean("Keystore file does not exist.  Create file? ")) {
                                ksFile = null;
                            }
                        }
                    }
                }

                ksPassword = promptForInput("Keystore Password: ");

                try {
                    ks = getKeyStore(ksFile, ksPassword);
                    Enumeration<String> aliases = ks.aliases();
                    if (aliases.hasMoreElements()) {
                        System.out.println("The keystore contains the following aliases: ");
                    }
                    while (aliases.hasMoreElements()) {
                        System.out.println(aliases.nextElement());
                    }
                } catch (Exception e) {
                    System.out.println("There was an error loading your keystore.  Reason: " +
                            e.getMessage());
                    return;
                }

            }

            boolean changes = false;
            for (Certificate cert: certs) {
                if (cert instanceof X509Certificate) {
```

```
                X509Certificate x509 = (X509Certificate) cert;
                printCertificateDetails(x509);
                boolean importCert = promptForBoolean("Import this certificate? ");
                if (importCert) {
                    String alias =
                            promptForInput("Choose an alias [<enter> for " + host + "]: ").trim();
                    if (alias.length() == 0) {
                        alias = host;
                    }

                    try {
                        importCertInKeyStore(cert, ks, alias);
                        changes = true;
                    } catch (KeyStoreException e) {
                        System.out.println("There was an error importing this certificate.  Reason: " +
                                e.getMessage());
                    }
                }
            }
        }

        if (changes) {
            System.out.println("Now saving the keystore back to file...");
            try {
                saveKeyStore(ks, ksFile, ksPassword);
            } catch (Exception e) {
                System.out.println("Couldn't save the keystore back to disk. " +
                        "All changes will be lost. Reason: " + e.getMessage());
            }
        } else {
            System.out.println("No changes were made to the keystore.");
        }
    }

    private void printCertificateDetails(X509Certificate x509) {
        System.out.println();
        System.out.println("A certificate was found on the server with the following information:");
        System.out.println("============");
        System.out.println("Owner:      " + x509.getSubjectX500Principal());
        System.out.println("Issuer:     " + x509.getIssuerX500Principal());
        System.out.println("Serial #:   " + x509.getSerialNumber());
        System.out.println("Valid from: " + x509.getNotBefore() + " until: " + x509.getNotAfter());
        System.out.println("============");
        System.out.println();
    }

    String promptForInput(String prompt) throws IOException {
        System.out.print(prompt);
        return in.readLine();
    }

    boolean promptForBoolean(String prompt) throws IOException {
        while (true) {
            String val = promptForInput(prompt).toLowerCase().trim();
            String[] trues = new String[] {"true", "t", "yes", "y", "ok"};
            String[] falses = new String[] {"false", "f", "no", "n"};
            if (Arrays.asList(trues).contains(val)) {
                return true;
            } else if (Arrays.asList(falses).contains(val)) {
                return false;
            }
        }
    }

    public static void main(String[] args) throws IOException {
        CertificateUtility cu = new CertificateUtility();
        cu.interactiveSession();
    }
}
```