BEA WebLogic Workshop™ Help

Version 8.1 SP4
December 2004

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Table of Contents

ProxyClient Samples

This section contains source code for the following samples.

Samples Included in This Section

mazegen Samples

register Samples

WSSE Samples

mazegen Samples

This section contains source code for the following samples.

Samples Included in This Section

java_client Samples

Readme.html Sample

swing_client Samples

java_client Samples

This section contains source code for the following samples.

Samples Included in This Section

MazeClient.java Sample

MazeClient.java Sample

This topic inludes the source code for the MazeClient.java Sample.

Sample Location

This sample is located in the following directory in your WebLogic Workshop installation:

BEA_HOME/weblogic81/samples/workshop/SamplesApp/ProxyClient/mazegen/java_client/

Sample Source Code

```
package mazegen.java_client;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.DataInputStream;
import java.io.IOException;
import java.io.*;
import java.rmi.RemoteException;

// proxy classes are located in the weblogic.jws.proxies package.
import weblogic.jws.proxies.*;

/**
 * <p>This is an example of a web service client that uses the web service
 * proxy generated by WebLogic Server.  This example is a simple Java
 * console application that exercises the MazeGenerator web service and
 * displays mazes returned from the web service.</p>
 *
 * <p>See "MazeGenerator Sample" in the WebLogic Workshop documentation.</p>
 *
 * <p>This client accesses the web service via proxy classes found in
 * SamplesApp/APP-INF/lib/MazeGenerator.jar.  That JAR file is obtained by selecting
 * the "Java Proxy" link on a web service's Test View Overview Page.  If
 * the web service is hosted locally, the URL of the Overview Page is:
 * http://localhost:7001/samples/proxy/mazegen/MazeGenerator.jws</p>
 */
public class MazeClient
{

    private class InputParams
    {
        int rows;
        int cols;

        public InputParams()
        {
            rows = -1;
            cols = -1;
        }
    }

    /*
     * <p>Declare the proxy objects.  There is a main proxy object with the
     * same name as the web service but with "_Impl" appended, then
     * protocol-specific proxie classes that contain the actual method stubs
```

```
     * for the web service's operations.</p>
     *
     * <p>This web service supports SOAP over HTTP, so it
     * has a proxy with the name <web service>SOAP.</p>
     */
    MazeGenerator_Impl m_Proxy = null;
    MazeGeneratorSoap m_ProxySoap = null;

    /**
     * Utility method that prints prompts to System.out and gets input
     * values from System.in.
     */
    private InputParams getInputParams()
    {
        String input = null;
        int rows = 0;
        int columns = 0;

        char [] charBuf = new char[255];
        int charsRead = 0;

        InputParams returnedParams = null;
        InputParams params = new InputParams();

        BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));

        System.out.print("Enter rows:");
        try
        {
            input = stdin.readLine();
            rows = Integer.parseInt(input);
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
        }
        catch (NumberFormatException ex) {
            String msg = "Number of rows must be a valid integer (entered value: '" + input + "
            throw(new NumberFormatException(msg));
        }
        System.out.println(input);

        params.rows = rows;

        System.out.print("Enter columns:");
        try
        {
            input = stdin.readLine();
            columns = Integer.parseInt(input);
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
        }
        catch (NumberFormatException ex) {
            String msg = "Number of columns must be a valid integer (entered value: '" + input
            throw(new NumberFormatException(msg));
        }
        System.out.println(input);

        params.cols = columns;
```

```
        if( params.rows > 0 && params.cols > 0 )
        {
            returnedParams = params;
        }

        return returnedParams;
    }

    /*
     * Main work method.  Loops forever (until invalid input), calling
     * getInputParams to collect input (desired number of rows and
     * columns) and then calling the web service to get a maze of the
     * desired size.
     */
    private void run()
    {
        InputParams params;

        try
        {
            /*
             * Instantiate the main proxy class. The proxy class has the same name as the
             * web service, with "_Impl" appended.
             *
             * Note that there is an alternative constructor in which you can pass the URL of
             * the target service's WSDL (which is typically the URL of the service with ?WSDL
             * appended).  That technique is used for "dynamic" proxies. The static WSDL is
             * already built into the generated proxy and will be used if the no-argument
             * proxy constructor is used (as below). Using the dynamic form causes an extra
             * hit on the server for every web service request submitted.
             */
            m_Proxy = new MazeGenerator_Impl();
        }
        catch (IOException ex)
        {
            System.out.println("Error getting proxy");
            ex.printStackTrace();
        }
        /*
         * Get the protocol-specific proxy class.
         */
        m_ProxySoap = m_Proxy.getMazeGeneratorSoap();

        while(true)
        {
            // get the desired number of rows and columns
            params = getInputParams();
            if( params != null )
            {
                try
                {
                    // invoke the web service to get the maze

                    String sMaze = m_ProxySoap.printRandomMaze(params.rows,
                                                               params.cols);

                    // print the response
                    System.out.println(sMaze);
                }
                catch (RemoteException ex)
```

```
                {
                    System.out.println("Error invoking service via proxy");
                    ex.printStackTrace();
                }
            }
            else
            {
                System.out.println("MazeClient: Invalid maze size parameters entered");
            }
        }

    }

    public static void main(String [] args)
    {

        MazeClient client = new MazeClient();
        client.run();

        return;
    }
}
```

Readme.html Sample

This topic inludes the source code for the Readme.html Sample.

Sample Location

This sample is located in the following directory in your WebLogic Workshop installation:

BEA_HOME/weblogic81/samples/workshop/SamplesApp/ProxyClient/mazegen/

Sample Source Code

```
<html>

<head>
<title>WebLogic Workshop Web Service Client Proxy Examples</title>
<style type="text/css">h1, h2, h3, h4, p, li, blockquote { font−family: Verdana, Arial, Helveti
p, li        { font−size: 12 }
blockquote   { font−size: 12 }
blockquote   { margin−left: 1em }
ul           { list−style−type: square }
</style>
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
</head>

<body>

<h1>WebLogic Workshop Web Service Java Client Proxy Examples</h1>
<h2>Introduction</h2>
<p>The samples in this project demonstrate how to call the proxy for a WebLogic
Workshop web service from a Java client. Two sample clients are provided in this
project: a standalone Java console application and a Swing application. A third
example demonstrates a JSP client using the web service proxy; that sample is
available in the WebApp project in the SamplesApp application, in
/WebApp/jspProxyClient/mazegen.</p>
<p>The sample web service, MazeGenerator.jws, is available in the SamplesApp
application in /WebServices/proxy/mazegen/. This is the web service that was
used to generate the proxy classes called by the client applications. The web
service exposes two operations. Each accepts two integers as parameters
representing the desired number of rows and columns in a randomly generated
maze. getRandomMaze returns the maze as an array of integers that encode the
walls present in each cell of the maze. printRandomMaze returns a &amp;quot;text
graphics&amp;quot; representation of the maze.</p>
<p>The samples access the web service through the web service proxy. The proxy
for any given Workshop web service is available from that service's Test View on
the Overview Page. The proxy comes as a web service−specific JAR file containing
proxy classes. To use the proxy, you also need the generic webserviceclient.jar
that is also available from Test View.</p>
<p>These examples require use of <b>ant</b>. ant is delivered with WLS and can
be found in BEA_HOME/weblogic81/server/bin. Please make sure that directory is
on your system path.</p>
<h2>Contents</h2>
<blockquote>
  <p>java_client/</p>
  <blockquote>
    <b>build.xml</b>: ant script that compiles the MazeClient standalone Java
```

```
    console client.<br>
    <b>MazeClient.java</b>: Standalone Java client of MazeGenerator web service,
    uses MazeGenerator proxy class.<br>
    <b>run.bat</b> | <b>run.sh</b>: Script that runs the client.
  </blockquote>
</blockquote>
<blockquote>
  <p>swing_client/</p>
  <blockquote>
    <b>build.xml</b>: ant script that compiles and runs <tt>MazeGUIClient</tt>.<br>
    <b>MazeGUIClient.java</b>: a rudimentary Swing app that uses the
    MazeGenerator web service via the proxy to specify maze parameters and
    display the resulting text maze. This client will be updated to display the
    maze graphically.<br>
  </blockquote>
  <p>JSP Clients</p>
  <blockquote>
    Two JSP clients are available at SamplesApp/WebApp/handlingData/traditional_vs_pageFlow_web
  </blockquote>
</blockquote>
<h2>Instructions</h2>
<h3>The Standalone Java Client</h3>
<ol>
  <li>In a cmd shell, cd to <tt>BEA_HOME/weblogic81/samples/workshop/SamplesApp/ProxyClient/maz
  <li>Run <b><tt>ant compile</tt></b> to compile the Java client. (<b>ant</b>
    must be on your PATH; <b>ant</b> can be found in BEA_HOME/weblogic81/server/bin)</li>
  <li>There is a <b><tt>run.bat</tt></b> (Windows) or <b><tt>run.sh</tt></b>
    (Linux/Unix) script to run the Java client. It relies on WL_HOME being set
    to BEA_HOME\weblogic81 on Windows and BEA_HOME/weblogic81 on
    Linux/Unix. Type <tt>run</tt> to run the client.</li>
  <li>Enter integers for rows and columns.</li>
</ol>
<p>You should see the maze printed as text.</p>
<p>The program will loop until you enter an invalid value (blank line or
non−numeric input) for the number of rows; then it will exit with an
exception.</p>
<h3>The Swing Client</h3>
<ol>
  <li>In a cmd shell, cd to <tt>BEA_HOME/weblogic81/samples/workshop/SamplesApp/ProxyClient/maz
  <li>Run <b><tt>ant run</tt></b> to compile and run the Swing client.</li>
  <li>Enter integers for rows and columns.</li>
  <li>Press the &amp;quot;Generate Maze&amp;quot; button.</li>
</ol>
<p>You should see the maze printed as text in the large text area.</p>
</body>
</html>
```

swing_client Samples

This section contains source code for the following samples.

Samples Included in This Section

MazeGUIClient.java Sample

MazeGUIClient.java Sample

This topic inludes the source code for the MazeGUIClient.java Sample.

Sample Location

This sample is located in the following directory in your WebLogic Workshop installation:

BEA_HOME/weblogic81/samples/workshop/SamplesApp/ProxyClient/mazegen/swing_client/

Sample Source Code

```
package mazegen.swing_client;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

import java.io.IOException;
import java.rmi.RemoteException;

// proxy classes are located in the weblogic.jws.proxies package.
import weblogic.jws.proxies.*;

/**
 * <p>This is an example of a web service client that uses the web wervice
 * proxy generated by WebLogic Server.  This example is a simple Swing
 * client that exercises the MazeGenerator web service and displays mazes
 * returned from the web service.</p>
 *
 * <p>See "MazeGenerator Sample" in the WebLogic Workshop documentation.</p>
 *
 * <p>This client accesses the web service via proxy classes found in
 * SamplesApp/APP-INF/lib/MazeGenerator.jar.  That JAR file is obtained by selecting
 * the "Java Proxy" link on a web service's Test View Overview Page.  If
 * the web service is served from the local machine, the URL to the Overview
 * Page is:
 * http://localhost:7001/samples/proxy/mazegen/MazeGanerator.jws</p>
 */
public class MazeGUIClient {
    private int rows = 11;
    private int cols = 26;

    JTextArea textArea;

    /*
     * <p>Declare the proxy objects.  There is a main proxy object with the
     * same name as the web service but with "_Impl" appended, then
     * protocol-specific proxie classes that contain the actual method
     * stubs for the web service's operations.</p>
     *
     * <p>This web service supports SOAP over HTTP, so it
     * has a proxy with the name <web service>SOAP.</p>
     */
    MazeGenerator_Impl proxy;
    MazeGeneratorSoap soapProxy;
```

```
/**
 * Utility function to create the Swing UI.
 *
 * Note that the anonymous ActionListener for generateButton
 * invokes the proxy with the number of desired rows and columns
 * and displays the response in textArea.
 */
public Component createComponents() {
    GridBagLayout gridbag = new GridBagLayout();
    GridBagConstraints constraints = new GridBagConstraints();

    /*******************************************************
     * create input panel
     */
    final JLabel rowLabel = new JLabel("Rows: ");
    final JTextField rowTextField = new JTextField(10);
    rowTextField.setColumns(4);
    rowTextField.setText("11");
    rowTextField.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            String text = rowTextField.getText();
            try
            {
                rows = Integer.parseInt(text);
            }
            catch (NumberFormatException ex)
            {
                // if the value isn't valid, default to 11
                rows = 11;
                rowTextField.setText("11");
            }
        }
    });
    rowTextField.addFocusListener(new FocusListener() {
        public void focusGained(FocusEvent e) {
        }
        public void focusLost(FocusEvent e) {
            String text = rowTextField.getText();
            try
            {
                rows = Integer.parseInt(text);
            }
            catch (NumberFormatException ex)
            {
                // if the value isn't valid, default to 11
                rows = 11;
                rowTextField.setText("11");
            }
        }
    });
    final JLabel colLabel = new JLabel("Columns: ");
    final JTextField colTextField = new JTextField(10);
    colTextField.setColumns(4);
    colTextField.setText("26");
    colTextField.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            String text = colTextField.getText();
            try
            {
                cols = Integer.parseInt(text);
            }
```

```
        catch (NumberFormatException ex)
        {
            // if the value isn't valid, default to 26
            cols = 26;
            colTextField.setText("26");
        }
    }
});
colTextField.addFocusListener(new FocusListener() {
    public void focusGained(FocusEvent e) {
    }
    public void focusLost(FocusEvent e) {
        String text = colTextField.getText();
        try
        {
            cols = Integer.parseInt(text);
        }
        catch (NumberFormatException ex)
        {
            // if the value isn't valid, default to 26
            cols = 26;
            colTextField.setText("26");
        }
    }
});

JButton generateButton = new JButton("Generate Maze");
generateButton.setMnemonic(KeyEvent.VK_G);
generateButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String mazeText;
        try
        {
            /*
             * Invoke the web service via the proxy.  Call
             * the web service's printRandomMaze method with
             * the desired number of rows and columns.  Display
             * the String response in textArea.
             */
            mazeText = soapProxy.printRandomMaze(rows, cols);
            textArea.setText(mazeText);
        }
        catch (RemoteException ex)
        {
            System.out.println("Error invoking service via proxy");
            ex.printStackTrace();
        }
    }
});

constraints.fill = GridBagConstraints.NONE;
constraints.gridx = GridBagConstraints.RELATIVE;
constraints.gridy = 0;
gridbag.setConstraints(generateButton, constraints);

JPanel inputPanel = new JPanel();
inputPanel.setBorder(BorderFactory.createEmptyBorder(
                                5, //top
                                5, //left
                                5, //bottom
                                5) //right
```

```
                                    );
        inputPanel.setLayout(new GridLayout(1,0,5,0));
        inputPanel.add(rowLabel);
        inputPanel.add(rowTextField);
        inputPanel.add(colLabel);
        inputPanel.add(colTextField);
        inputPanel.add(generateButton);

        constraints.fill = GridBagConstraints.HORIZONTAL;
        constraints.gridx = 0;
        constraints.gridy = 0;
        gridbag.setConstraints(inputPanel, constraints);

        /*********************************************************
         * create output panel
         */
        JPanel outputPanel = new JPanel();

        textArea = new JTextArea(25, 80);
        textArea.setFont(new Font("Courier", Font.PLAIN, 12));

        outputPanel.add(textArea);

        constraints.fill = GridBagConstraints.HORIZONTAL;
        constraints.gridx = 0;
        constraints.gridy = GridBagConstraints.RELATIVE;
        gridbag.setConstraints(outputPanel, constraints);

        /*********************************************************
         * create main panel
         */
        JPanel mainPanel = new JPanel();
        mainPanel.setBorder(BorderFactory.createEmptyBorder(
                                    10, //top
                                    10, //left
                                    10, //bottom
                                    10) //right
                                    );
        mainPanel.setLayout(gridbag);
        mainPanel.add(inputPanel);
        mainPanel.add(outputPanel);

        return mainPanel;
    }

    /**
     * Utility function to instantiate the proxy, then get the
     * SOAP-specific proxy from the main proxy.
     */
    private void CreateProxy()
    {
        try
        {
            /*
             * Construct the proxy.
             *
             * Note that there is an alternative constructor in which you can pass the URL of
             * the target service's WSDL (which is typically the URL of the service with ?WSDL
             * appended).  That technique is used for "dynamic" proxies. The static WSDL is
             * already built into the generated proxy and will be used if the no-argument
             * proxy constructor is used (as below). Using the dynamic form causes an extra
```

```
             * hit on the server for every web service request submitted.
             */
            proxy = new MazeGenerator_Impl();
            soapProxy = proxy.getMazeGeneratorSoap();
        }
        catch (IOException ex)
        {
            System.out.println("Error creating proxy");
        }
    }

    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel(
                UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) {}

        // Create the top-level container
        JFrame frame = new JFrame("MazeGUIClient");
        MazeGUIClient app = new MazeGUIClient();

        // Create the proxy
        app.CreateProxy();

        // Create the UI
        Component contents = app.createComponents();
        frame.getContentPane().add(contents, BorderLayout.CENTER);

        /*
         * Make the UI visible, then implicitly loop forever on
         * user events until the program is terminated.
         */
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }
}
```

register Samples

This section contains source code for the following samples.

Samples Included in This Section

Readme.html Sample

RegisterClient.java Sample

Readme.html Sample

This topic inludes the source code for the Readme.html Sample.

Sample Location

This sample is located in the following directory in your WebLogic Workshop installation:

BEA_HOME/weblogic81/samples/workshop/SamplesApp/ProxyClient/register/

Sample Source Code

```
<html>
<head>
<title>
RegisterClient.java: WebLogic Workshop Web Service Client Proxy Example
</title>
<style type="text/css">
h1,
h2,
h3,
h4,
p,
li,
blockquote
{
        font−family: Verdana, Arial, Helvetica, sans−serif;
        margin−left: 0px;
}
h1 { font−size: 150%; }
h2 { font−size: 130%; }
p, li { font−size: 12; }
blockquote { font−size: 12; }
blockquote { margin−left: 1em; }
ul { list−style−type: square; }
</style>
</head>
<body>
<h1>RegisterClient.java: WebLogic Workshop Web Service Client Proxy Example</h1>
<h2>Introduction</h2>
<p>
This sample demonstrates how to create a Java client for a Workshop web service.
In particular, it demonstrates how to interact with a conversational web service and how to man
compound Java types used in the web service's interface. RegisterClient.java is a client for th
defined in WL_HOME/samples/workshop/SamplesApp/WebServices/proxy/register/RegisterPerson.jws.</
<p>RegisterClient.java is a standalone Java application that accesses the web service through t
web service proxy.  The proxy for any given WebLogic
Workshop web service is available from that service's Test View on the Overview page by clickin
the <b>Java Proxy</b> link.  The
proxy classes are packaged in a web service−specific JAR file.  To use the proxy, you
also need the generic webserviceclient.jar that you can obtain by clicking the <b>Proxy Support
Overview page of a web service's Test View.
link.</p>

<a name="Environment"></a>
<h2>Environment</h2>
<p>In order to run this sample, the following must be true:</p>
```

```
<ul>

<li>
The <b>WL_HOME</b> environment variable must be set to BEA_HOME/weblogic81,
where BEA_HOME is the "bea" directory in which the WebLogic Platform
is installed.
</li>

<li>
<b>ant</b> must be on your PATH. <b>ant</b> is delivered with WLS and can
be found in WL_HOME/server/bin. Please make sure that
directory is on your PATH.
</li>

<li>
<b>java</b> must be on your PATH. <b>java</b> is delivered with WLS
and can be found in BEA_HOME/jdk141_03/bin. Please make sure that
directory is on your PATH.
</li>

</ul>

<h2>Contents</h2>

<blockquote>
  <b>build.xml</b>
<blockquote>
    ant script that compiles the RegisterClient standalone Java console client.<br>
</blockquote>
</blockquote>

<blockquote>
  <b>RegisterClient.java</b>
<blockquote>
    standalone Java application that is a client of RegisterPerson.jws. See comments in the fil
the details of writing a web service Java client.<br>
</blockquote>
</blockquote>

<blockquote>
  <b>RegisterPerson.jar</b>
<blockquote>
    client proxy JAR file containing the web service−specific JAX−RPC classes needed to access
was obtained from the <b>Java Proxy</b> link on the Overview page of Test View for RegisterPers
both the web service−specific JAR file and the generic <b>webserviceclient.jar</b> file that is
by clicking the <b>Proxy Support Jar</b> link.<br>
</blockquote>
</blockquote>

<blockquote>
  <b>run.bat/run.sh</b>
<blockquote>
    run scripts for the client for Windows and Linux/Unix, respectively. A run target in build.
ant steals stdin. These scripts may require minor modification for your environment.<br>
</blockquote>
</blockquote>

<h2>Compiling and Running RegisterClient.java</h2>
<p>The following instructions assume that the WL_HOME and PATH environment variables are set co
<a href="#Environment">Environment</a> section above.</p>
<ol>
```

```
<li>Make sure the Workshop Example Server is running.</li>
<li>In a cmd shell, cd to:
<ul>
<li>Linux/Unix: <tt>$WL_HOME/samples/workshop/SamplesApp/ProxyClient/register</tt></li>
<li>Windows: <tt>%WL_HOME%\samples\workshop\SamplesApp\ProxyClient\register</tt></li>
</ul>
</li>
<li>Type <b><tt>ant</tt></b> to compile the Java client.</li>
<li>Type <b><tt>run</tt></b> to run the RegisterClient Java client.
<li>Enter menu item numbers to perform the indicated actions. Each menu item causes invocation
service operation.</li>
</ol>

</body>
</html>
```

RegisterClient.java Sample

This topic inludes the source code for the RegisterClient.java Sample.

Sample Location

This sample is located in the following directory in your WebLogic Workshop installation:

BEA_HOME/weblogic81/samples/workshop/SamplesApp/ProxyClient/register/

Sample Source Code

```
package register;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.rmi.RemoteException;
import java.util.Date;

// proxy classes are located in the weblogic.jws.proxies package.
import weblogic.jws.proxies.RegisterPerson_Impl;
import weblogic.jws.proxies.RegisterPersonSoap;

/**
 * Import the Person class.
 * Don't import the Contact data structure, because we're illustrating
 * use of both SOAP document/literal and SOAP-RPC encoded parameters, so
 * we declare them with fully qualified names as we use the classes below.
 */
import org.openuri.www.Person;

/**
 * <p>This is an example of a web service client that uses the web service
 * proxy generated by WebLogic Server.</p>
 *
 * <p>See "TBD Sample" in the WebLogic Workshop documentation.</p>
 *
 * <p>This client accesses the web service via proxy classes found in
 * WEB-INF/lib/RegisterPerson.jar.  That JAR file is obtained by selecting
 * the "Java Proxy" link on a web service's Test View Overview Page.  If
 * the web service is hosted locally, the URL of the Overview Page is:
 * http://localhost:7001/samples/proxy/register/RegisterPerson.jws</p>
 */
public class RegisterClient
{

    /*
     * <p>Declare the proxy objects.  There is a main proxy object with the
     * same name as the web service but with "_Impl" appended, then
     * protocol-specific proxy classes that contain the actual method stubs
     * for the web service's operations.</p>
     *
     * <p>This web service supports SOAP over HTTP, so it
     * has a proxy class with the name <web service>SOAP.</p>
     */
    RegisterPerson_Impl m_proxyImpl = null;
```

```
RegisterPersonSoap m_proxy = null;

boolean m_ConversationStarted = false;

/**
 * Utility method that prints prompts to System.out and gets input
 * values from System.in. Also validates input to make sure web service
 * is in proper conversational state for the requested action.
 */
private int getInput()
{
    String input = null;
    int menuItem = 0;
    boolean validSelectionMade = false;

    BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));

    while( !validSelectionMade )
    {
        System.out.println("_____________________");
        System.out.println("   1 - set person");
        System.out.println("   2 - get person");
        System.out.println("   3 - set home contact info to A");
        System.out.println("   4 - set home contact info to B");
        System.out.println("   5 - get home contact");
        System.out.println("   6 - set work contact info to A");
        System.out.println("   7 - set work contact info to B");
        System.out.println("   8 - get work contact");
        System.out.println("   9 - exit");
        System.out.println();

        System.out.print("Enter menu item number:");
        try
        {
            input = stdin.readLine();
            menuItem = Integer.parseInt(input);
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
            continue;
        }
        catch (NumberFormatException ex) {
            System.out.println("Please enter a valid menu item number:");
            continue;
        }

        /*
         * setPerson is a "start conversation" method. Make sure there isn't already
         * a conversation started.
         */
        if( menuItem == 1 )
        {
            if( !m_ConversationStarted )
            {
                validSelectionMade = true;
            }
            else
            {
                System.out.println("There is a current conversation already, " +
                                   "please choose another menu option.");
```

```
            }
        }

        /*
         * All methods except setPerson require that a conversation already exist.
         */
        else if( menuItem > 1 && menuItem < 9 )
        {
            if( m_ConversationStarted )
            {
                validSelectionMade = true;
            }
            else
            {
                System.out.println("There is no current conversation, " +
                                   "please choose setPerson (1) first.");
            }
         }
         else if( menuItem == 9 )
         {
            validSelectionMade = true;
         }

    }
    System.out.println();

    return menuItem;
}

/**
 * Utility function to print a Contact to stdout. Specifically, prints a Contact
 * that is SOAP document/literal encoded.
 */
public void printLiteralContact(org.openuri.www.Contact c)
{
    if( c != null )
    {
        System.out.println("      " + c.getStreet());
        System.out.print("      " + c.getCity());
        System.out.println(", " + c.getState() + " " + c.getZip());
    }
    else
    {
        System.out.println("      none");
    }
}

/**
 * Utility function that prints a Person to stdout.
 */
public void printPerson(Person p)
{
    if( p != null )
    {
        System.out.println("   " + p.getName());

        System.out.println("   home:");
        printLiteralContact(p.getHome());

        System.out.println("   work:");
        printLiteralContact(p.getWork());
```

```
    }
    else
    {
        System.out.println("   person is null");
    }
}

/**
 * Main work method.  Loops forever (until the "exit" menu item is selected),
 * calling getInput to collect input and then invoking the requested operation
 * on the web service.
 */
private void run()
{
    boolean done = false;
    int menuItem = 0;

    /*
     * Utility variables to hold values returned from web service methods
     */
    Person p;
    org.openuri.www.Contact soapLiteralContact;
    org.openuri.www.encodedTypes.Contact soapRPCContact;

    /*
     * Initialized variables used in the setPerson, setHomeContact and setWorkContact
     * operations. ContactA is only ever used in SOAP document/literal. ContactB is used
     * in SOAP document/literal form only when passed to setPerson. The "set xxx contact
     * to B" operations always use the SOAP-RPC versions of the operation and Contact
     * class for illustration.
     */
    org.openuri.www.Contact soapLiteralContactA =
        new org.openuri.www.Contact("100 A Street", "Aville", "AA", "11111", "111-111-1111"
    org.openuri.www.Contact soapLiteralContactB =
        new org.openuri.www.Contact("200 B Street", "Bville", "BB", "22222", "222-222-2222"
    org.openuri.www.encodedTypes.Contact soapRPCContactB =
        new org.openuri.www.encodedTypes.Contact("200 B Street", "Bville", "BB", "22222", "

    try
    {
        /*
         * Instantiate the main proxy class. The proxy class has the same name as the
         * web service, with "_Impl" appended.
         *
         * Note that there is an alternative constructor in which you can pass the URL of
         * the target service's WSDL (which is typically the URL of the service with ?WSDL
         * appended).  That technique is used for "dynamic" proxies. The static WSDL is
         * already built into the generated proxy and will be used if the no-argument
         * proxy constructor is used (as below). Using the dynamic form causes an extra
         * hit on the server for every web service request submitted.
         */
        m_proxyImpl = new RegisterPerson_Impl();
    }
    catch (IOException ex)
    {
        System.out.println("Error getting proxy");
        ex.printStackTrace();
    }
    /*
     * Get the protocol-specific proxy class.
     */
```

```
m_proxy = m_proxyImpl.getRegisterPersonSoap();

while( !done )
{
    // get the menu item and perform the action
    menuItem = getInput();

    switch( menuItem )
    {
        case 1:
            System.out.println("set person");

            p = new Person("John Doe", soapLiteralContactA, soapLiteralContactB);

            /*
             * Invoke the web service's setPerson method by invoking the stub method
             * of the proxy class.
             *
             * All proxy class stub methods declare that they throw RemoteException,
             * so we have to surround all calls to proxy stub methods in try/catch
             * blocks.
             */
            try
            {
                m_proxy.setPerson(p);
                m_ConversationStarted = true;
            }
            catch(RemoteException ex)
            {
                System.out.println("error calling setPerson");
                ex.printStackTrace();
            }

            break;
        case 2:
            System.out.println("get person");

            p = null;

            /*
             * Invoke the web service's getPerson method.
             */
            try
            {
                p = m_proxy.getPerson();
            }
            catch(RemoteException ex)
            {
                System.out.println("error calling getPerson");
                ex.printStackTrace();
            }

            printPerson(p);

            break;
        case 3:
            System.out.println("set home contact info to A");

            /*
             * Invoke the web service's setHomeContact method.
             *
```

```
             * For illustration purposes, the target web service defines
             * both SOAP document/literal and SOAP-RPC versions of the
             * setHomeContact and setWorkContact methods. This action uses
             * the document/literal version.
             */
            try
            {
                m_proxy.setHomeContact(soapLiteralContactA);
            }
            catch(RemoteException ex)
            {
                System.out.println("error calling setHomeContact");
                ex.printStackTrace();
            }

            break;
        case 4:
            System.out.println("set home contact info to B");

            /*
             * Invoke the web service's setHomeContactRPC method.
             *
             * For illustration purposes, the target web service defines
             * both SOAP document/literal and SOAP-RPC versions of the
             * setHomeContact and setWorkContact methods. This action uses
             * the SOAP-RPC version.
             */
            try
            {
                m_proxy.setHomeContactRPC(soapRPCContactB);
            }
            catch(RemoteException ex)
            {
                System.out.println("error calling setHomeContactRPC");
                ex.printStackTrace();
            }

            break;
        case 5:
            System.out.println("get home contact");

            /*
             * Invoke the web service's getHomeContact method.
             */
            try
            {
                soapLiteralContact = m_proxy.getHomeContact();
                printLiteralContact(soapLiteralContact);
            }
            catch(RemoteException ex)
            {
                System.out.println("error calling getHomeContact");
                ex.printStackTrace();
            }

            break;
        case 6:
            System.out.println("set work contact info to A");

            /*
             * Invoke the web service's setWorkContact method.
```

```
             *
             * For illustration purposes, the target web service defines
             * both SOAP document/literal and SOAP-RPC versions of the
             * setHomeContact and setWorkContact methods. This action uses
             * the document/literal version.
             */
            try
            {
                m_proxy.setWorkContact(soapLiteralContactA);
            }
            catch(RemoteException ex)
            {
                System.out.println("error calling setWorkContact");
                ex.printStackTrace();
            }

            break;
        case 7:
            System.out.println("set work contact info to B");

            /*
             * Invoke the web service's setWorkContactRPC method.
             *
             * For illustration purposes, the target web service defines
             * both SOAP document/literal and SOAP-RPC versions of the
             * setHomeContact and setWorkContact methods. This action uses
             * the SOAP-RPC version.
             */
            try
            {
                m_proxy.setWorkContactRPC(soapRPCContactB);
            }
            catch(RemoteException ex)
            {
                System.out.println("error calling setWorkContactRPC");
                ex.printStackTrace();
            }

            break;
        case 8:
            System.out.println("get work contact");

            /*
             * Invoke the web service's getWorkContact method.
             */
            try
            {
                soapLiteralContact = m_proxy.getWorkContact();
                printLiteralContact(soapLiteralContact);
            }
            catch(RemoteException ex)
            {
                System.out.println("error calling getWorkContact");
                ex.printStackTrace();
            }

            break;
        case 9:
            System.out.println("exit");

            /*
```

```
                     * Invoke the web service's endSession method, which terminates the
                     * conversation. This client program also exits.
                     */
                    if( m_ConversationStarted )
                    {
                        try
                        {
                            m_proxy.endSession();
                            m_ConversationStarted = false;
                        }
                        catch(RemoteException ex)
                        {
                            System.out.println("error calling endSession");
                            ex.printStackTrace();
                        }
                    }

                    done = true;
                    break;
                }
            }
        }

        public static void main(String [] args)
        {

            RegisterClient client = new RegisterClient();
            client.run();

            return;
        }
}
```

WSSE Samples

This section contains source code for the following samples.

Samples Included in This Section

clientCert Samples

Readme.html Sample

token Samples

clientCert Samples

This section contains source code for the following samples.

Samples Included in This Section

KeyUtil.java Sample

MyCompanyClient.java Sample

KeyUtil.java Sample

This topic inludes the source code for the KeyUtil.java Sample.

Sample Location

This sample is located in the following directory in your WebLogic Workshop installation:

BEA_HOME/weblogic81/samples/workshop/SamplesApp/ProxyClient/WSSE/clientCert/

Sample Source Code

```
package WSSE.clientCert;

import java.io.IOException;
import java.io.FileInputStream;
import java.security.KeyStore;
import java.security.PrivateKey;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.X509Certificate;
import java.security.cert.CertificateException;

public class KeyUtil {

    public static KeyStore loadKeystore(String filename, String password)
        throws KeyStoreException, IOException, NoSuchAlgorithmException, CertificateException {
    final KeyStore ks = KeyStore.getInstance("JKS");
    ks.load(new FileInputStream(filename), password.toCharArray());
    return ks;
    }

    public static PrivateKey getPrivateKey(String alias, String password, KeyStore keystore)
        throws Exception {
    PrivateKey result =
        (PrivateKey) keystore.getKey(alias, password.toCharArray());

    return result;
    }

    public static X509Certificate getCertificate(String alias, KeyStore keystore)
        throws Exception {
    X509Certificate result = (X509Certificate) keystore.getCertificate(alias);
    return result;
    }

}
```

MyCompanyClient.java Sample

This topic inludes the source code for the MyCompanyClient.java Sample.

Sample Location

This sample is located in the following directory in your WebLogic Workshop installation:

BEA_HOME/weblogic81/samples/workshop/SamplesApp/ProxyClient/WSSE/clientCert/

Sample Source Code

```
package WSSE.clientCert;

import java.util.List;
import java.util.ArrayList;
import java.security.KeyStore;
import java.security.PrivateKey;
import java.security.cert.X509Certificate;
import java.security.cert.CertificateException;
import javax.xml.rpc.ServiceException;
import javax.xml.namespace.QName;
import javax.xml.rpc.handler.HandlerInfo;
import javax.xml.rpc.handler.HandlerRegistry;
import weblogic.Home;
import weblogic.webservice.context.WebServiceSession;
import weblogic.webservice.context.WebServiceContext;
import weblogic.webservice.core.handler.WSSEClientHandler;
import weblogic.webservice.WLMessageContext;
import weblogic.xml.security.wsse.Security;
import weblogic.xml.security.wsse.Token;
import weblogic.xml.security.wsse.BinarySecurityToken;
import weblogic.xml.security.wsse.SecurityElementFactory;
import weblogic.xml.security.specs.SignatureSpec;
import weblogic.jws.proxies.MyCompany;
import weblogic.jws.proxies.MyCompany_Impl;
import weblogic.jws.proxies.MyCompanySoap;
import weblogic.xml.security.UserInfo;
import weblogic.xml.security.specs.EncryptionSpec;

/*
 * This client (1) adds a digital certificate to the invoking SOAP message,
 *             (2) encrypts the invoking SOAP message,
 *             and (3) adds a username/password token to the SOAP message.
 */
public class MyCompanyClient {

    public static final String USERNAME="weblogic";
    public static final String USER_PASSWORD="weblogic";

    /*
     * Home.getPath() returns the String value WL_HOME + "/server".
     * Home.getPath().replaceAll("/server", "") extracts the value of WL_HOME.
     */
    private static final String CLIENT_KEYSTORE = Home.getPath().replaceAll("/server", "") + "/
    private static final String KEYSTORE_PASS = "password";
    private static final String KEY_ALIAS = "client1";
```

```
    private static final String SERVER_KEY_ALIAS = "mycompany";
    private static final String KEY_PASSWORD = "password";

    public static void main(String[] args) {

        try{
            /*
             * Instantiate the main proxy class. The proxy class has the same name as the
             * web service, with "_Impl" appended.
             */
            MyCompany myservice = new MyCompany_Impl("http://localhost:7001/WebServices/securit

            WebServiceContext context = myservice.context();
            WebServiceSession session = context.getSession();

            /**
             * Registers a handler for the SOAP message traffic.
            HandlerRegistry registry = myservice.getHandlerRegistry();
            List list = new ArrayList();
            list.add(new HandlerInfo(WSSEClientHandler.class, null, null));
            registry.setHandlerChain(new QName("hello"), list);
*/
            final KeyStore keystore = KeyUtil.loadKeystore(CLIENT_KEYSTORE, KEYSTORE_PASS);

         // Add a client certificate

            X509Certificate clientcert = KeyUtil.getCertificate(KEY_ALIAS, keystore);

            PrivateKey clientprivate = KeyUtil.getPrivateKey(KEY_ALIAS, KEY_PASSWORD, keystore)

            SecurityElementFactory factory = SecurityElementFactory.getDefaultFactory();

            Token x509token = factory.createToken(clientcert, clientprivate);

            SignatureSpec sigSpec = SignatureSpec.getDefaultSpec();

            Security security = factory.createSecurity(null);

            security.addSignature(x509token, sigSpec);

            security.addToken(x509token);

        // Encrypts the SOAP body

            X509Certificate servercert = KeyUtil.getCertificate(SERVER_KEY_ALIAS, keystore);

            EncryptionSpec encSpec = EncryptionSpec.getDefaultSpec();

            Token serverToken = factory.createToken(servercert, null);

            security.addEncryption(serverToken, encSpec);

        // Adds a username/password token

            /**
             * Set the username and password token for SOAP message sent from the client, throu
             * the proxy, to the web service.
             */
            UserInfo ui = new UserInfo("weblogic", "weblogic");
            session.setAttribute(WSSEClientHandler.REQUEST_USERINFO, ui);
```

```
            /**
             * Adds the username / password token to the SOAP header.
             */
            Security security2 = factory.createSecurity(null);
            security.addToken(ui);
            session.setAttribute(WSSEClientHandler.REQUEST_SECURITY, security);

            /*
             * Get the protocol-specific proxy class.
             */
            MyCompanySoap msg = myservice.getMyCompanySoap();

            /*
             * Add the security element to the request.
             */
            context.getSession().setAttribute(WSSEClientHandler.REQUEST_SECURITY, security);

            /*
             * Set the client's private key to decrypt the response
             */
            session.setAttribute(WSSEClientHandler.KEY_ATTRIBUTE, clientprivate);

            /**
             * Invoke the web service method hello()
             */
            String result = msg.hello();

            System.out.println();
            System.out.println("Web Service Response:");
            System.out.println(result);
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

Readme.html Sample

This topic inludes the source code for the Readme.html Sample.

Sample Location

This sample is located in the following directory in your WebLogic Workshop installation:

BEA_HOME/weblogic81/samples/workshop/SamplesApp/ProxyClient/WSSE/

Sample Source Code

```
<html>

<head>
<title>WebLogic Workshop Web Service Client Proxy Examples</title>
<style type="text/css">h1, h2, h3, h4, p, li, blockquote { font-family: Verdana, Arial, Helveti
p, li        { font-size: 12 }
blockquote   { font-size: 12 }
blockquote   { margin-left: 1em }
ul           { list-style-type: square }
</style>
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
</head>

<body>

<h1>Calling a WSSE-enabled Web Service Through Its Java Proxy Classes</h1>
<h2>Introduction</h2>
<p>These samples show how to call a WSSE-enabled web service through the web service's
Java proxy classes.</p>
<p>Two web service clients are included: MyCompanyClient.java and WebServiceBClient.java.</p>
<p>The target web services are included in the SamplesApp
application at /WebServices/security/wsse/reqResp/mycompany/MyCompany.jws and
WebServices/security/wsse/usertoken/webServiceB/WebServiceB.jws.
<h3>WebServiceB.jws</h3>
<p>The web service WebServiceB exposes one operation: hello(), which returns the string "Hello,
<p>Any client to this web service
must provide a username/password token in the header of the SOAP message requests.
<h3>MyCompany.jws</h3>
<p>The web service MyCompany.jws exposes one operation: hello(), which returns the string "Hell
<p>Any client to this web service
must (1) provide a digital certificate, (2) encrypt SOAP messages requests, and
(3) provide a username/password token in the header of the SOAP message requests.
<h3>Proxy Classes</h3>
<p>The samples access the web service through the web service proxy. The proxy
for any given Workshop web service is available from that service's Test View on
the Overview Page. The proxy is packaged as a web service-specific JAR file containing
proxy classes. To use the proxy, you also need the generic webserviceclient.jar
that is also available from Test View.</p>
<p>These examples require use of <b>ant</b>. ant is delivered with WLS and can
be found in BEA_HOME/weblogic81/server/bin. Please make sure that directory is
on your PATH.</p>
<h2>Contents</h2>
<blockquote>
  <p>WSSE/clientCert</p>
```

```
  <blockquote>
    <b>build.xml</b>: ant script that compiles the MyCompanyClient standalone Java
    console client.<br>
    <b>MyCompanyClient.java</b>: Standalone Java client of the MyCompany web service,
    uses MyCompany proxy class.<br>
    <b>run.bat</b>: Script that runs the Java console client.
  </blockquote>
  <p>WSSE/token</p>
  <blockquote>
    <b>build.xml</b>: ant script that compiles the MyWebServiceBClient standalone Java
    console client.<br>
    <b>MyWebServiceBClient.java</b>: Standalone Java client of the MyWebServiceB web service,
    uses MyWebServiceB proxy class.<br>
    <b>run.bat</b>: Script that runs the Java console client.
  </blockquote>
</blockquote>
<h2>Instructions for Running The Standalone Java Clients</h2>
<h3>Running MyCompanyClient.java</h3>
<ol>
  <li>In a cmd shell, cd to <tt>BEA_HOME/weblogic81/samples/workshop/SamplesApp/ProxyClient/WSS
  <li>Run <b><tt>ant compile</tt></b> to compile the Java client. (<b>ant</b>
    must be on your PATH; <b>ant</b> can be found in BEA_HOME/weblogic81/server/bin)</li>
  <li>There is a <b><tt>run.bat</tt></b> (Windows) script to run the Java client.
  It relies on WL_HOME being set
    to BEA_HOME\weblogic81 on Windows.  Type <tt>run</tt> to run the client.</li>
</ol>
<p>You should see the message returned by the MyCompany web service.

<h3>Running MyWebServiceB.java</h3>
<ol>
  <li>In a cmd shell, cd to <tt>BEA_HOME/weblogic81/samples/workshop/SamplesApp/ProxyClient/WSS
  <li>Run <b><tt>ant compile</tt></b> to compile the Java client. (<b>ant</b>
    must be on your PATH; <b>ant</b> can be found in BEA_HOME/weblogic81/server/bin)</li>
  <li>There is a <b><tt>run.bat</tt></b> (Windows) script to run the Java client.
  It relies on WL_HOME being set
    to BEA_HOME\weblogic81 on Windows.  Type <tt>run</tt> to run the client.</li>
</ol>
<p>You should see the message returned by the WebServiceB web service.</p>
</body>
</html>
```

token Samples

This section contains source code for the following samples.

Samples Included in This Section

WebServiceBClient.java Sample

WebServiceBClient.java Sample

This topic inludes the source code for the WebServiceBClient.java Sample.

Sample Location

This sample is located in the following directory in your WebLogic Workshop installation:

BEA_HOME/weblogic81/samples/workshop/SamplesApp/ProxyClient/WSSE/token/

Sample Source Code

```
package WSSE.token;

import java.util.ArrayList;
import java.util.List;
import javax.xml.namespace.QName;
import javax.xml.rpc.handler.HandlerInfo;
import javax.xml.rpc.handler.HandlerRegistry;
import weblogic.jws.proxies.*;
//import weblogic.jws.proxies.WebServiceB;
//import weblogic.jws.proxies.WebServiceB_Impl;
//import weblogic.jws.proxies.WebServiceBSoap;
import weblogic.webservice.context.WebServiceContext;
import weblogic.webservice.context.WebServiceSession;
import weblogic.webservice.core.handler.WSSEClientHandler;
import weblogic.xml.security.UserInfo;
import weblogic.xml.security.wsse.Security;
import weblogic.xml.security.wsse.SecurityElementFactory;

public class WebServiceBClient
{

    public static void main(String[] args){

        try{
            /*
             * Instantiate the main proxy class. The proxy class has the same name as the
             * web service, with "_Impl" appended.
             */
            WebServiceB myservice = new WebServiceB_Impl("http://localhost:7001/WebServices/sec

            WebServiceContext context = myservice.context();
            WebServiceSession session = context.getSession();

            /**
             * Registers a handler for the SOAP message traffic.
             */
            HandlerRegistry registry = myservice.getHandlerRegistry();
            List list = new ArrayList();
            list.add(new HandlerInfo(WSSEClientHandler.class, null, null));
            registry.setHandlerChain(new QName("hello"), list);

            /**
             * Set the username and password token for SOAP message sent from the client, throu
             * the proxy, to the web service.
             */
```

```
            UserInfo ui = new UserInfo("weblogic", "weblogic");
            session.setAttribute(WSSEClientHandler.REQUEST_USERINFO, ui);

            /**
             * Adds the username / password token to the SOAP header.
             */
            SecurityElementFactory factory = SecurityElementFactory.getDefaultFactory();
            Security security = factory.createSecurity(null);
            security.addToken(ui);
            session.setAttribute(WSSEClientHandler.REQUEST_SECURITY, security);

            /*
             * Get the protocol-specific proxy class.
             */
            WebServiceBSoap msg=myservice.getWebServiceBSoap();

            /**
             * Invoke the web service method hello()
             */
            String result=msg.hello();

            System.out.println();
            System.out.println("Web Service Response:");
            System.out.println(result);
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }

}
```