Message-level security with JAX-WS on WebSphere Application Server V7: Integrating JEE authorization

Skill Level: Intermediate

Henry Chung (johndoe@us.ibm.com)
Author1 job title
Author1 company

27 Jan 2010

In Part 1, you learned how to provide message level security using JAX-WS on WebSphere Application Server V7, including how to use policy sets to encrypt and sign messages, and how to use a UsernameToken profile for authentication. In Part 2, you'll learn how to use the UsernameToken passed in the SOAP header as the JEE principal to provide programmatic authorization in the service provider.

Introduction

Web Services Security (WS-Security) is an OASIS standard that describes how to implement message-level security with Web services. Specifically, WS-Security describes how to add confidentiality (such as encryption), integrity (such as digital signatures), and propagate security tokens for authentication (such as username and password) in SOAP messages. However, the WS-Security specification allows sending multiple security tokens simultaneously in the SOAP message, and typically Java™ Platform, Enterprise Edition (JEE) Web services provider implementation performs authorization checks based on the principal (identity) from one of the security tokens. In this article, we'll describe how to configure WebSphere to select which security token of the SOAP message as a JEE principal that can be used for authorization decisions.

Note that the JEE security model supports declarative security authorization as well as programmatic security for both Web containers and EJB containers. There are subtle differences between using the Web container programmatic APIs (such as

`getUserPrincipal()`) and the EJB container programmatic APIs (such as `getCallerPrincipal()`). However, the scope of this article is to discuss how to configure Web services in order to specify that one of the tokens in the SOAP header should be used as the JEE principal. Once this principal has been set, you can simply use the JEE security model and WebSphere Base Security APIs as you normally would.

You can use the JEE security model for authorization either declaratively or programmatically for both servlets and EJBs. However, for the purposes of this article, we'll demonstrate a servlet-based Web service that uses the programmatic JEE APIs to get the principal. You can extend the sample to use the JEE programmatic APIs to perform programmatic authorization checks in servlet-based base Web service providers or configure JEE role-based method-level security for EJB. JEE declarative and programmatic security for the Web container as well as the EJB container is covered in other materials, and are not the focus of this article. (See Resources for more information.) Our goal is to demonstrate how to enable the integration of the message-level security tokens for use with the JEE authorization framework on WebSphere Application Server.

Create a JAX-WS service provider

1. Using Rational Application Developer (Application Developer) V7.5.2, create a new dynamic Web project with a project name of `HelloWorldProject`.

2. Next, create a new Java class with the name `HelloWorldProvider` and copy the contents of Listing 1 into this new class.
   Listing 1. HelloWorldProvider.java

```
package com.ibm.dwexample;
import javax.annotation.Resource;
import javax.jws.WebService;
import javax.xml.ws.WebServiceContext;

@WebService
public class HelloWorldProvider {
  @Resource WebServiceContext wsCtx;

  public String sayHello(String msg) {
    System.out.println("[provider] received " + msg);
    System.out.println("[provider] user = " + wsCtx.getUserPrincipal());
    return "Hello " + msg;
  }
}
```

   The interesting part of the HelloWorldProvider code is the `@Resource WebServiceContext`. This line allows the JAX-WS runtime to inject the Web service context and enables you to access the JEE principal from

the context. However, in order for this code to actually return the correct principal in Application Server, you must configure the Caller in the service provider binding; otherwise, you may get a result of "Principal: /UNAUTHENTICATED*quot;.

3. Right-click the **HelloWorldProject** and select **Run As => Run on Server**. Ensure that **Run server with resources on Server** is selected in the **Publishing settings for WebSphere Application Server** section.

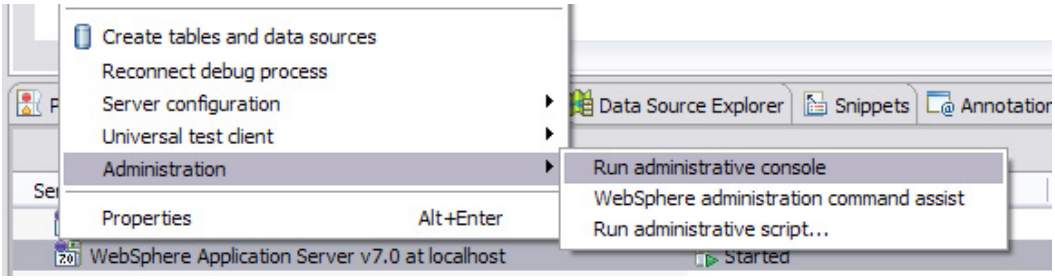
4. Select a **WebSphere Application Server v7.0** server profile and click **Finish**.

Secure the service provider

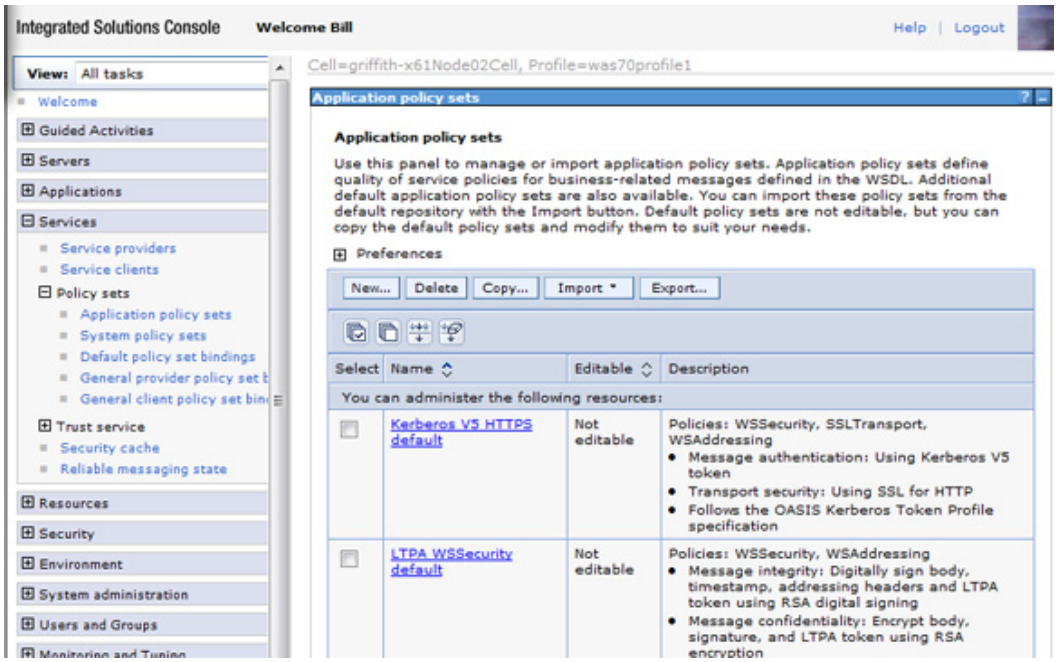
Policy sets and policy set bindings are covered in Part 1, so we'll go straight into creating a policy set that we'll use to specify a UsernameToken as the authentication token for the Web service. Once this policy set has been created and attached to the service provider, you'll create a server-side binding in which you'll specify which token will be used as the primary security token--that is, the JEE principal. You need to do this because the WS-Security specification allows attaching multiple tokens for authentication, thus additional metadata is required to identify which is the primary security token. In WebSphere, this metadata is known as the Caller and is configured as part of the binding for WS-Security, as we'll show in this article.

We'll use the Application Server administrative console to create the policy set, attach the policy set to your service provider, and create the binding by which this service provider will adhere.

1. From Application Developer, right-click the Application Server V7 runtime in the **Servers** view and select **Administration => Run administrative console** as shown in Figure 1.
Figure 1. Launch the administrative console



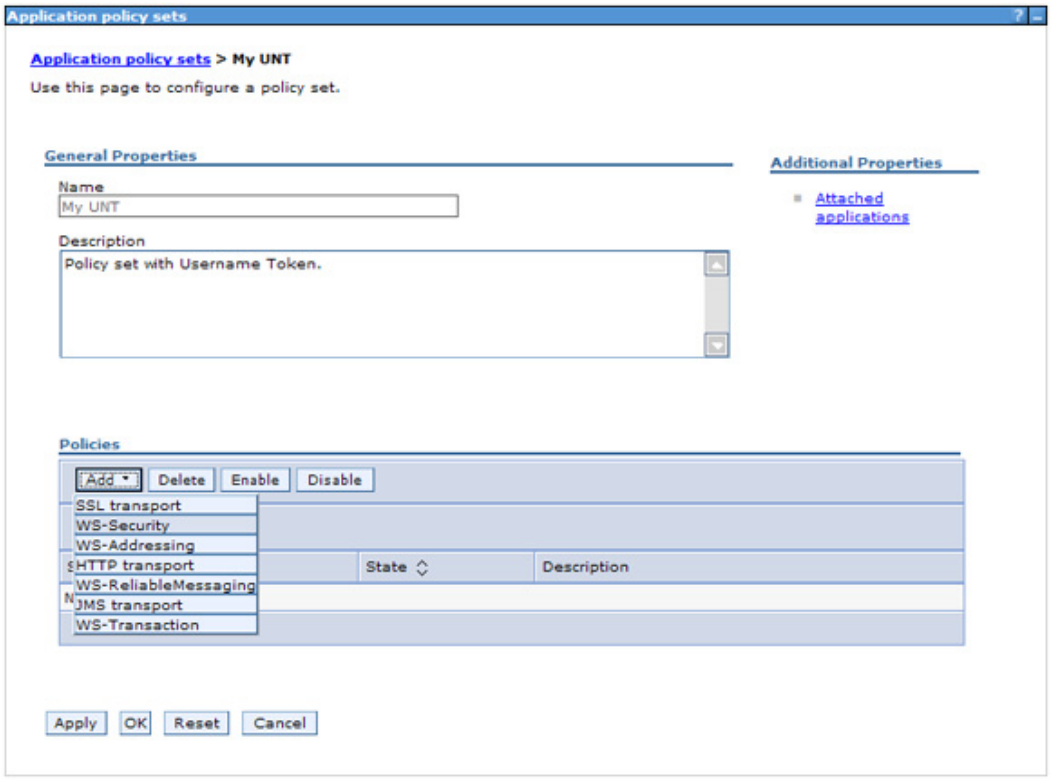
2. From the administrative console, select **Services => Policy sets => Application policy sets** as shown in Figure 2.
Figure 2. Application policy sets



3. Click **New** to create a new policy set.

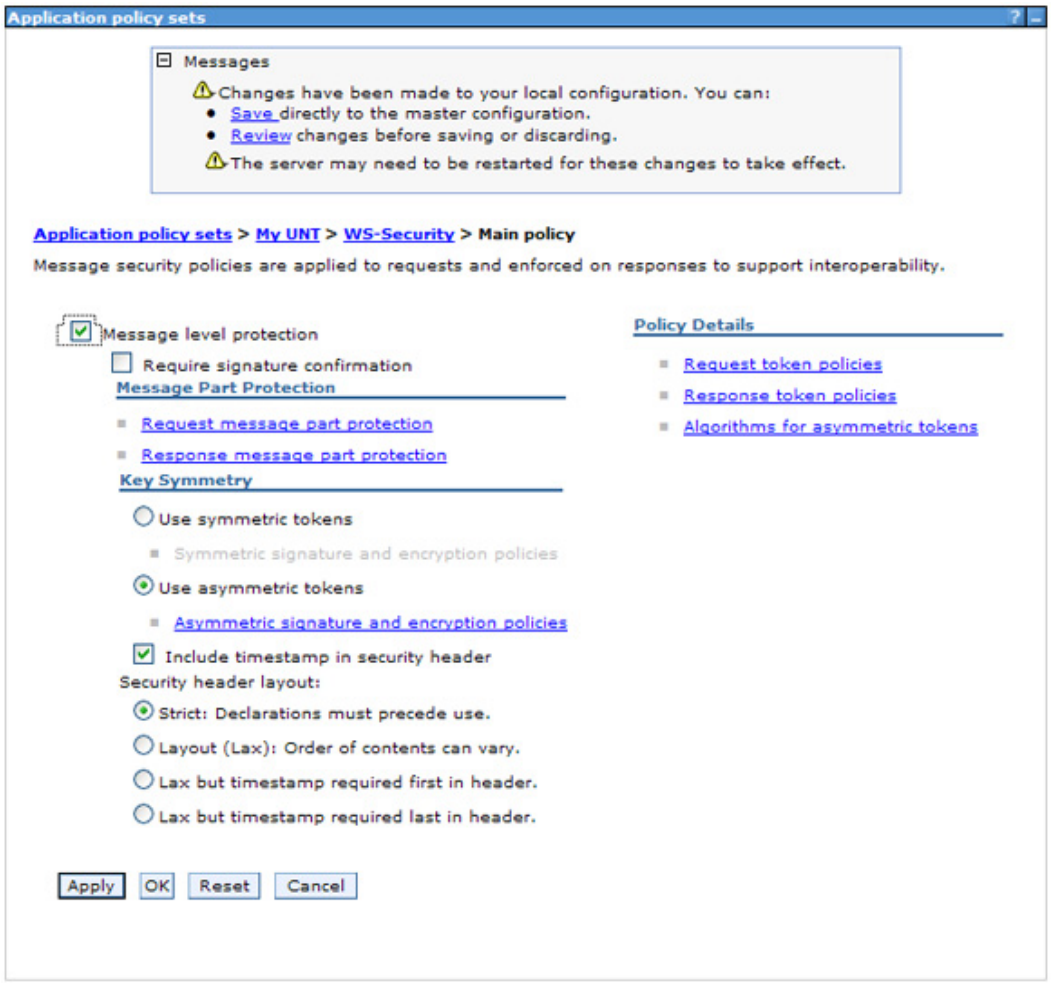
4. Specify `My UNT` as the name for the new policy set and add a description in the **Description** field, then click **Apply** .

5. Next click **Add** in the **Policies** section and choose **WS-Security** as the policy to be added as shown in Figure 3.
Figure 3. New policy set



6. Once the policy has been added to your new policy set, simply click **WS-Security** to configure it.

7. Click **Main policy**; you should see a screen that looks like Figure 4.
Figure 4. Configure WS-Security policy



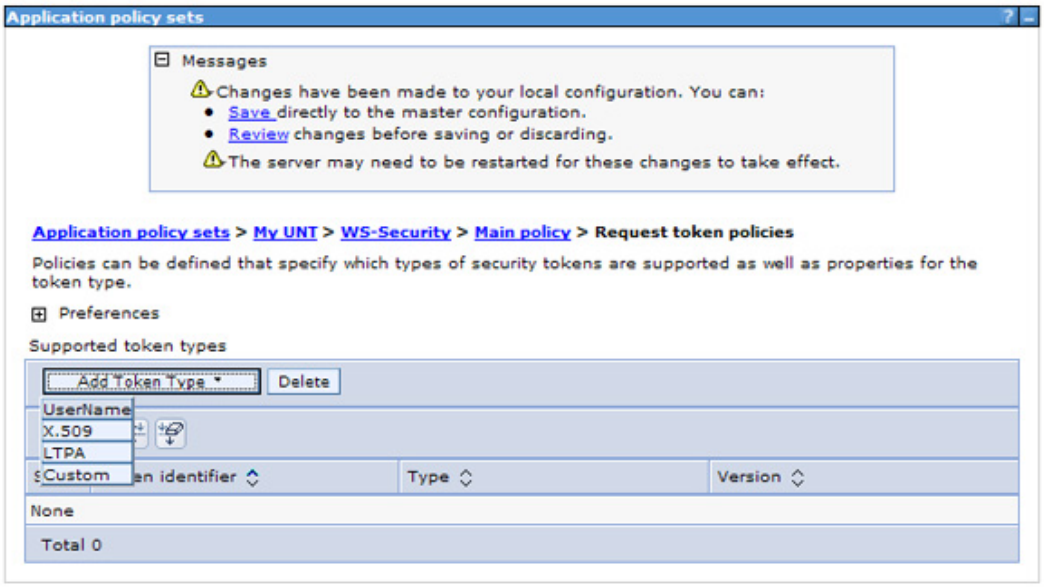
8. By default, the WS-Security policy is created with message-level protection, as described in Part 1. However, in order to simplify things for this article, disable message-level protection by unchecking **Message level protection**, then clicking **Apply**.

9. Since our policy requires a UsernameToken to extract the JEE principal, you need to add a UsernameToken to the WS-Security policy by doing the following:

    1. Click **Request token policies** in the **Policy Details** section of the Main Policy.

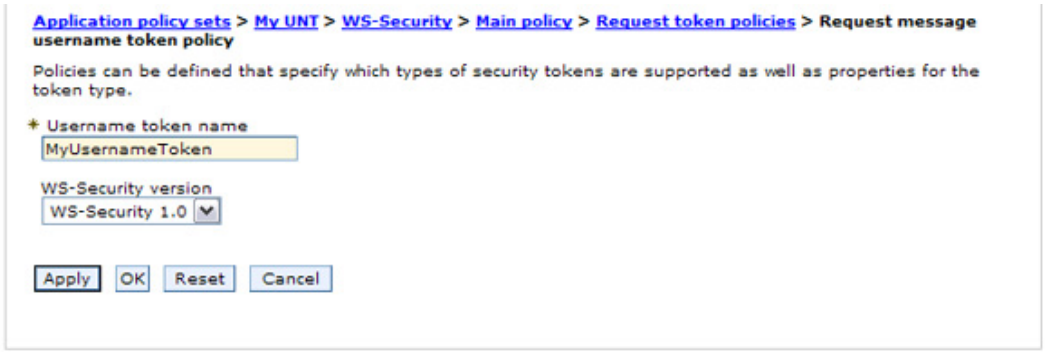
    2. Click **Add Token Type** and choose **UserName** as shown in Figure 5.

Figure 5. Add UsernameToken to Request Token policy



10. Specify `MyUsernameToken` for the **Username token** and leave **WS-Security 1.0** as the WS-Security version as shown in Figure 6, then click **Apply**.

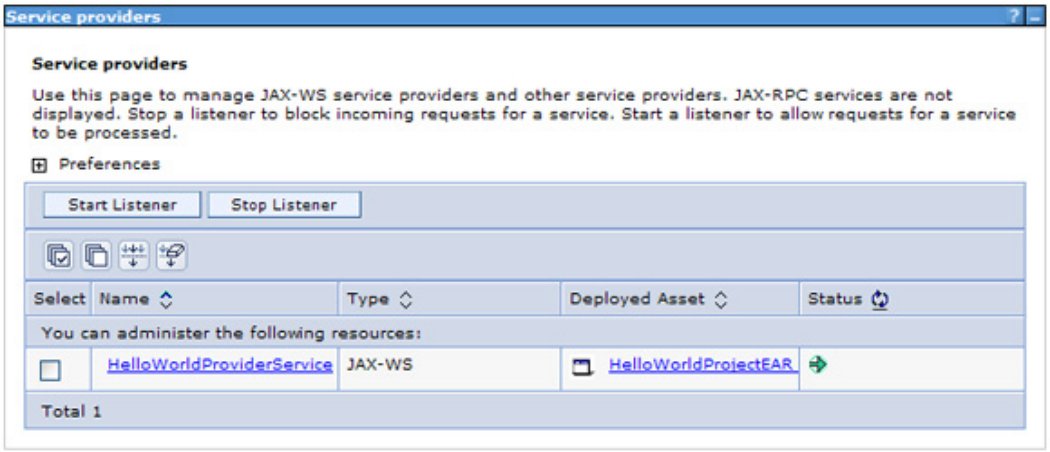
Figure 6. Specify UsernameToken



11. Click **Save** to save the changes directly to the master configuration. You should see a screen that looks like Figure 7.

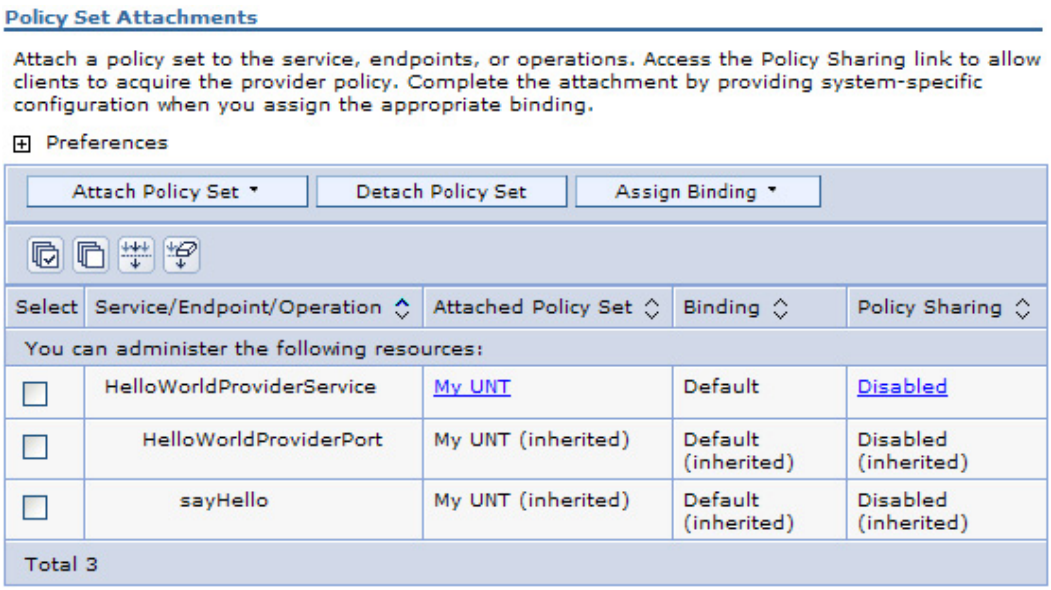
Figure 7. Configured UsernameToken policy

12. Now that the policy set is created, you need to attach it to your service provider. From the administrative console, select **Services => Service providers** to get the list of JAX-WS service providers, and select **HelloWorldProviderService**, as shown in Figure 8.
Figure 8. JAX-WS service providers



13. Check **HelloWorldProviderService**, click **Attach Policy Set**, and select your policy set (for example, **My UNT**).

14. The My UNT policy set is now attached to the HelloWorldProviderService, as shown in Figure 9.
Figure 9. Attach policy set to service provider

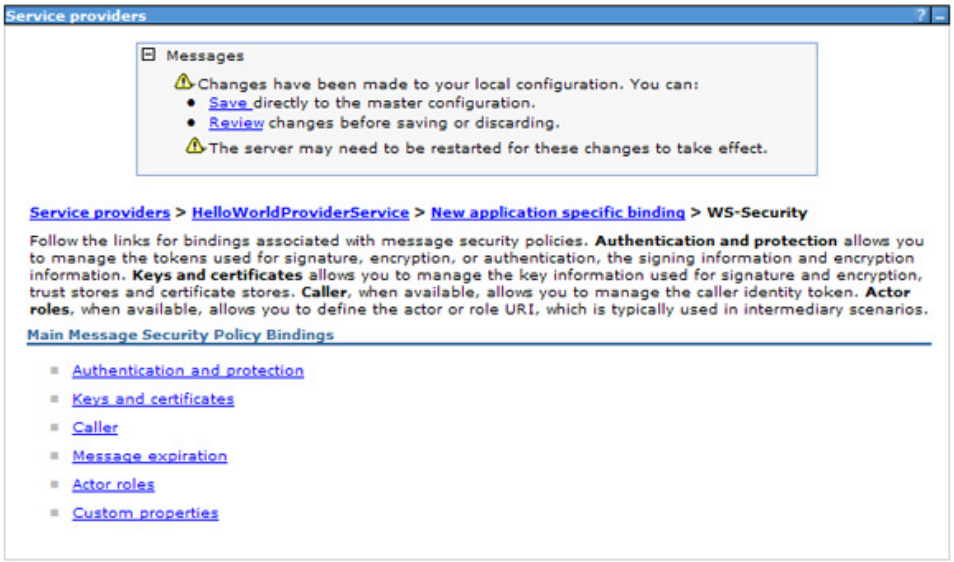


15. The policy set specifies the "what," while the bindings specify the "how." Therefore, you need to configure policy set bindings for this service provider. To do this:

    a. Check **HelloWorldProviderService**, then click **Assign Binding** and select **New Application Specific Binding**.

    b. Specify `ServerUNTBinding` for **Bindings configuration name**, then click **Add** and select **WS-Security** to create the application specific binding as shown in Figure 10.

Figure 10. Policy set binding configuration



Note that the binding assignment checked the policy set to determine which policies needed to be configured. In this case, the

policy set contained a WS-Security policy, which is why this policy was included in the **Add** drop-down menu.

16. Since the WS-Security policy set that you added to your service provider includes the `UsernameToken` as a required token of the requester, you need to specify the "how" for this policy in the binding by doing the following:

    a. Display the details of **ServerUNTBinding** by clicking **Authentication and protection**.

    b. Navigate to the authentication tokens section and click **request:MyUsernameToken**. You should see a screen like Figure 11.
    Figure 11. UsernameToken identity
    

    c. Keep the default values for this scenario, and click **OK**.

17. You now have specified that the UsernameToken to be passed in the SOAP header according to the WS-Security specification is to be used as the authentication token by the service provider. However, remember that the WS-Security specification allows more than one token to be passed in the request message, so now you'll need to specify to WebSphere which of these tokens is to be used in creating the WebSphere credentials (in other words, the JEE subject), so that the identity of the specific token can be used for JEE security, such as role-based authorization checking. In WebSphere, this is done by configuring the caller as follows:

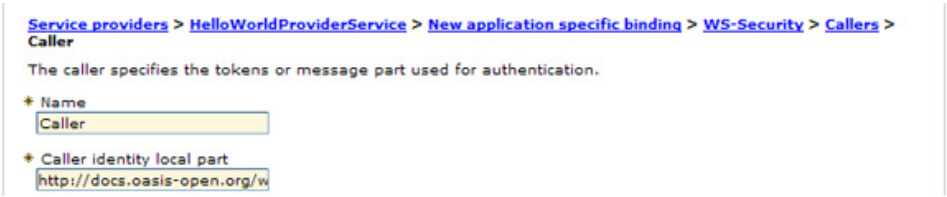
    a. Click **Caller** (see Figure 10) from the Callers dialog, then click **New** .

    b. Enter `Caller` for the **Name**.

    c. Enter `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-user` for the **Caller identity local part**, as shown in Figure 12. Note that this URL is the value of the **Local part** of the authentication token

shown in Figure 11.

Figure 12. Specify caller



d. Click **OK** to accept this caller, then click **Save** to save this binding to the master configuration.

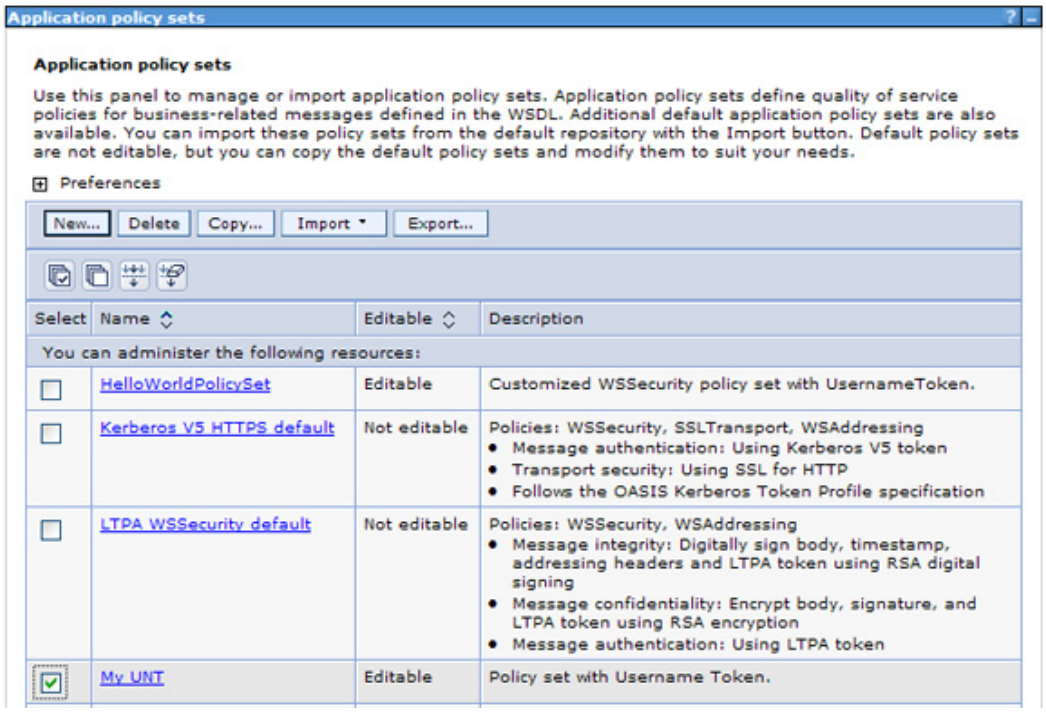
Consume the secure service

Perhaps the easiest way to ensure that the service consumer adheres to the policies of the service provider is to use the same policy set. You can do this by exporting the service provider policy set from the Application Server administrative console, then importing it into Application Developer.

To export the policy set, do the following:

1. From the administrative console, select **Services => Policy sets => Application policy sets**.
2. Check the **My UNT** policy set, then click **Export** =>, as shown in Figure 13.

Figure 13. Export the policy set



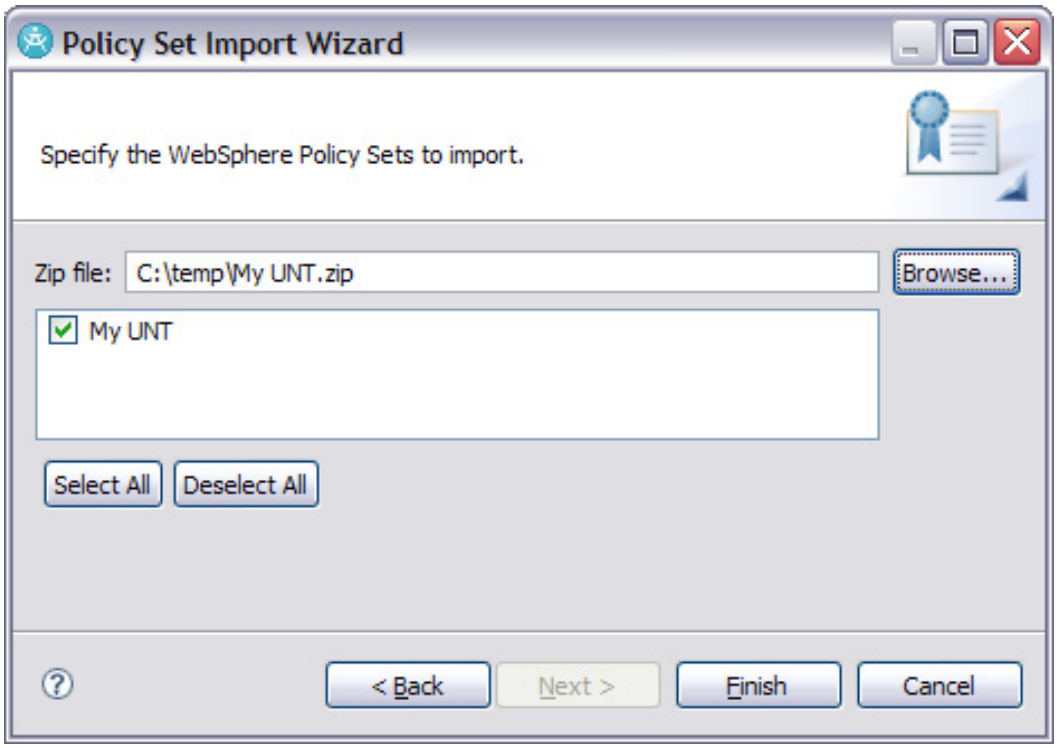
3. Click **My UNT.zip** and save the file somewhere on your local drive; for example, c:\temp.

4. Click **OK** to save the file.

To import the policy set into Application Developer, do the following:

1. From the Application Developer main menu, select **File => Import => Web services => WebSphere Policy Sets** , then click **Next**.

2. Click **Browse** and select the **My UNT.zip** file that you exported above. The wizard reads the zip file and displays the policy sets included in it, as shown in Figure 14.
Figure 14. Import the policy set



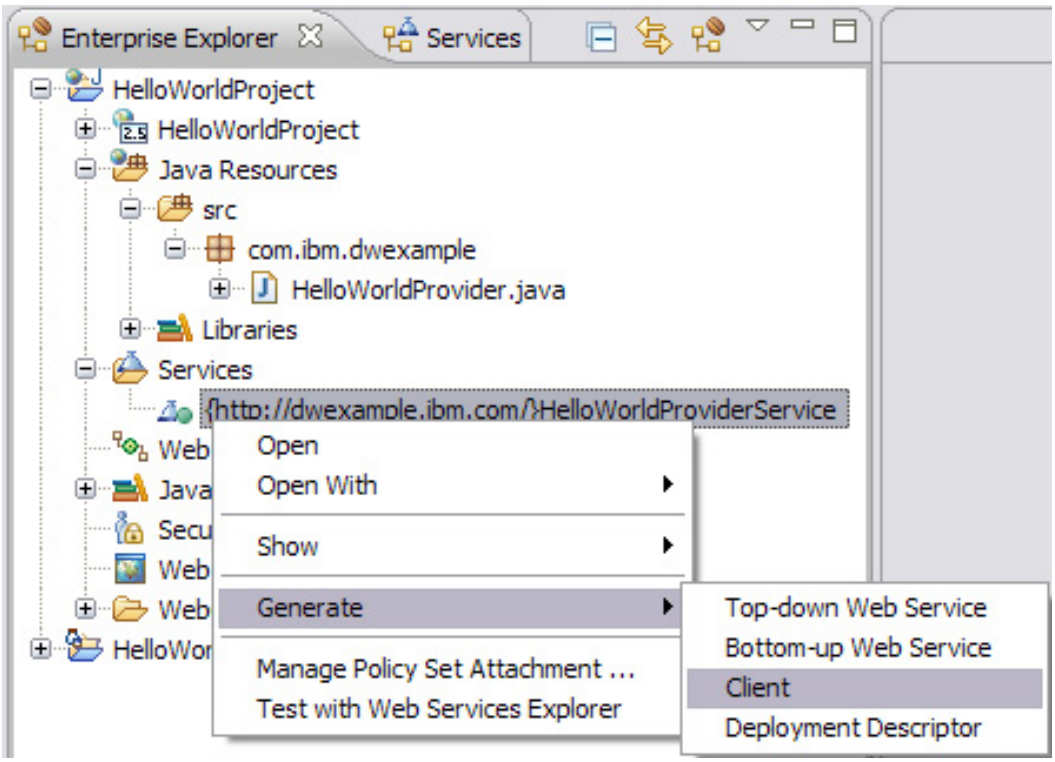
3. Ensure that **My UNT** is checked and click **Finish**.

Now that you've imported the policy set into Application Developer, you need to create a service consumer client to attach the policy set to:

1. In Application Developer, select **File => New => Other => Java => Java Project** to create a new Java project to hold the consumer.

2. Specify `HelloWorldConsumer` as the **Client project name**, then click **Finish**. If prompted to change to the Java perspective, click **No**.

3. Now select the service provider from which Application Developer will generate a client proxy and select **Generate => Client**, as shown in Figure 15.
Figure 15. Generate the JAX-WS client proxy



4. From the Web Service Client wizard, ensure that IBM WebSphere JAX-WS is the chosen Web service runtime, then click **Client project:**.

5. Specify `HelloWorldConsumer` as the **Client project name**, then choose **HelloWorldConsumer** as the client project and click **OK**.

6. Accept the defaults and click **Finish**. Application Developer will generate the JAX-WS client proxy class and supporting classes.

7. Right-click the generated **HelloWorldConsumer** project, and select **New => Class**.

8. Specify `com.ibm.dwexample` as the package name and `ClientTest` as the Java class name, then click **Finish**.

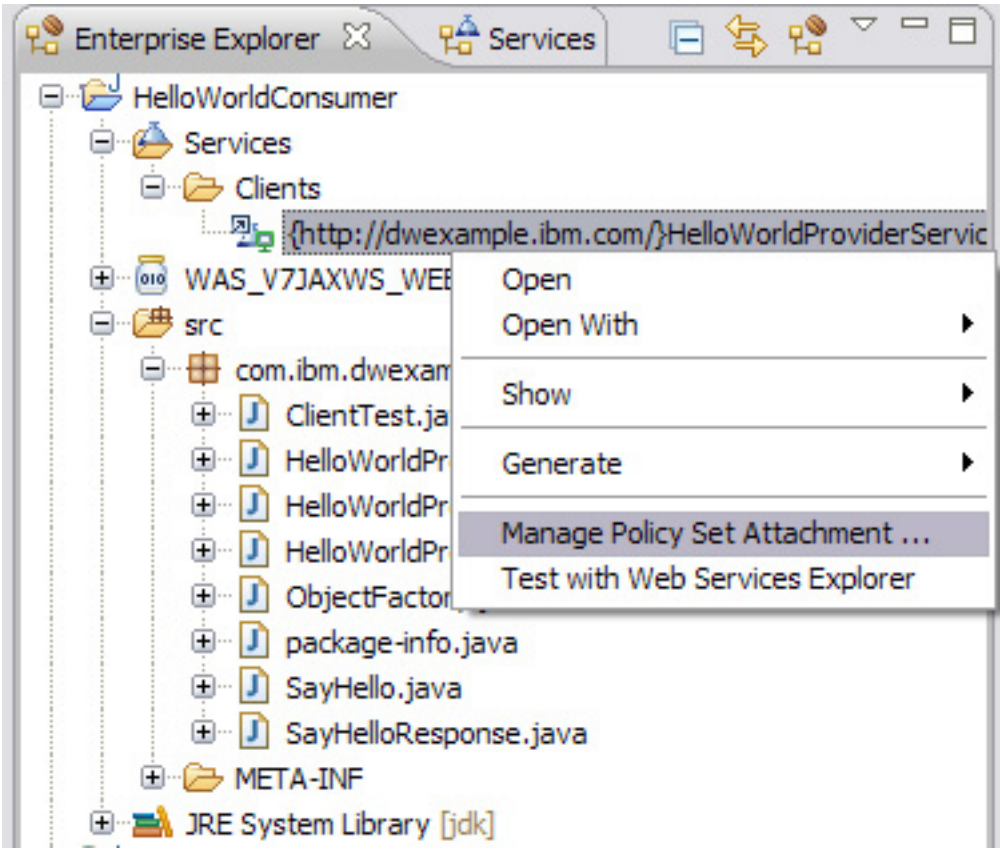
9. Replace the generated client code with the code in Listing 2 and save the file.
Listing 2. ClientTest.java

```
package com.ibm.dwexample;
import com.ibm.dwexample.HelloWorldProvider;
import com.ibm.dwexample.HelloWorldProviderService;

public class ClientTest {

  public static void main(String[] args) {
```

```
        HelloWorldProviderService srv = new HelloWorldProviderService();
        HelloWorldProvider port = srv.getHelloWorldProviderPort();

        String resp = port.sayHello("World");
        System.out.println("[response] " + resp);
    }
}
```

Now that you've created the JAX-WS consumer, you need to attach the imported policy set to the consumer, then generate a client-side policy set binding. To do this, complete the following steps:

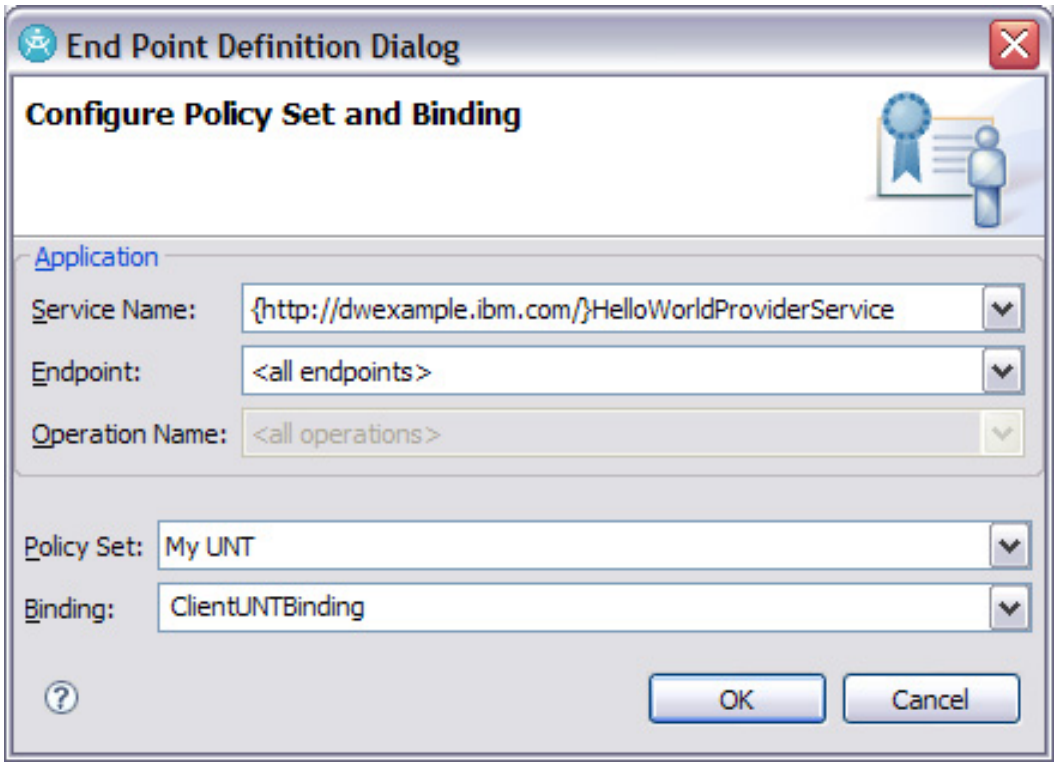
1. Navigate to the **HelloWorldConsumer** project and select **Services => Clients => {http://dwexample.ibm.com/}HelloWorldProviderService => Manage Policy Set Attachment** as shown in Figure 16.
   Figure 16. Manage policy set attachment



2. Click **Next**, then **Add**.

3. Verify that the service name is set to **{http://dwexample.ibm.com/}HelloWorldProviderService**, then select **Policy Set => My UNT**.

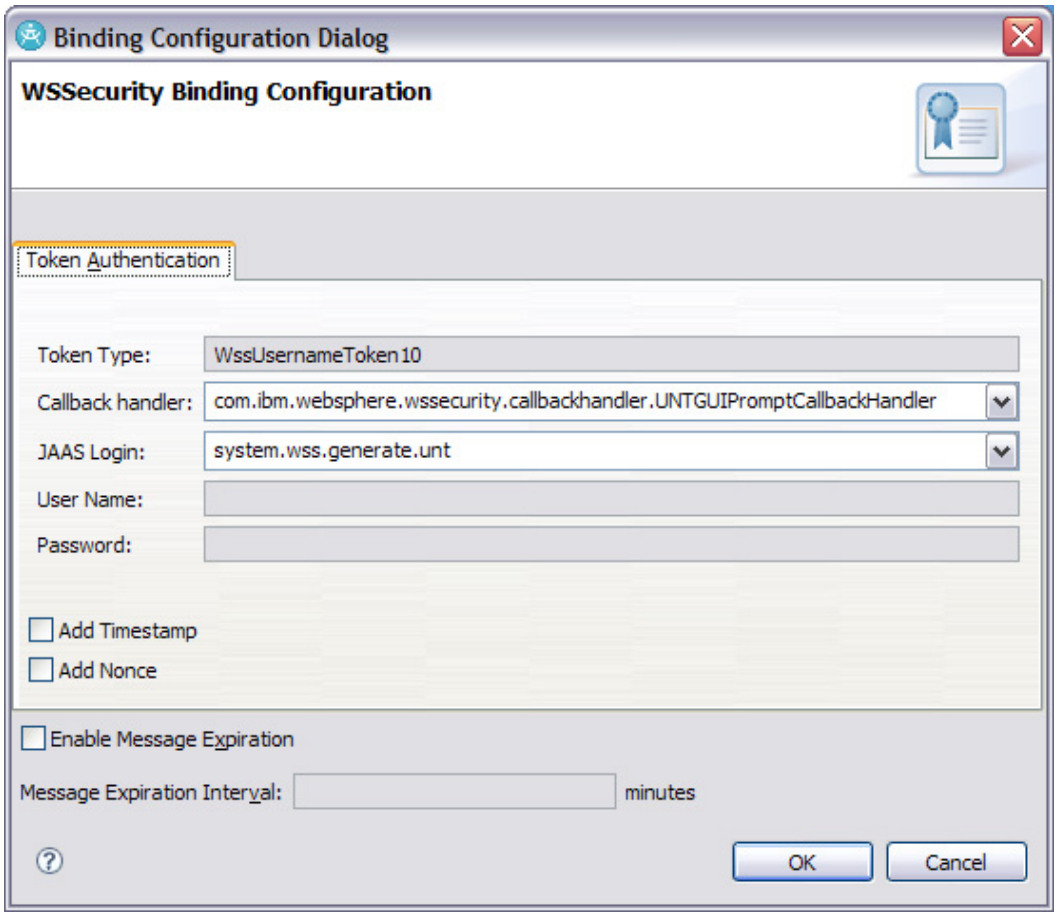
4. Enter `ClientUNTBinding` as the **Binding name**, and click **OK**, as shown in Figure 17.
Figure 17. Attach the policy set to the consumer



You've now attached the policy set that you created in Application Server and attached it to the JAX-WS consumer. You've also assigned the name to the client-side binding (`ClientUNTBinding`). The final step is to configure the binding:

1. Select the **WSSecurity** policy type in the bindings configuration and click **Configure**.

2. Select **com.ibm.websphere.wssecurity.callbackhandler.UNTGUIPromptCallbackHandler** as the callback handler, as shown in Figure 18.
Figure 18. JAX-WS consumer binding configuration



3. Click **OK**, then **Finish** .

You've now assigned a policy set and a corresponding policy set binding to the service consumer. You can now test the code to make sure it's really working.

Run the sample application

Because the code used in this article demonstrates using a UsernameToken (that is, a username and password in the SOAP header) as the authentication credentials for authenticating with Application Server, you need to ensure security is enabled on Application Server before you test. To do this, from the Application Server administrative console, ensure that **Enable administrative security** and **Enable application security** are both checked. If security was not enabled, you'll need to restart the Application Server for the security changes to take effect.

To test the application, do the following:

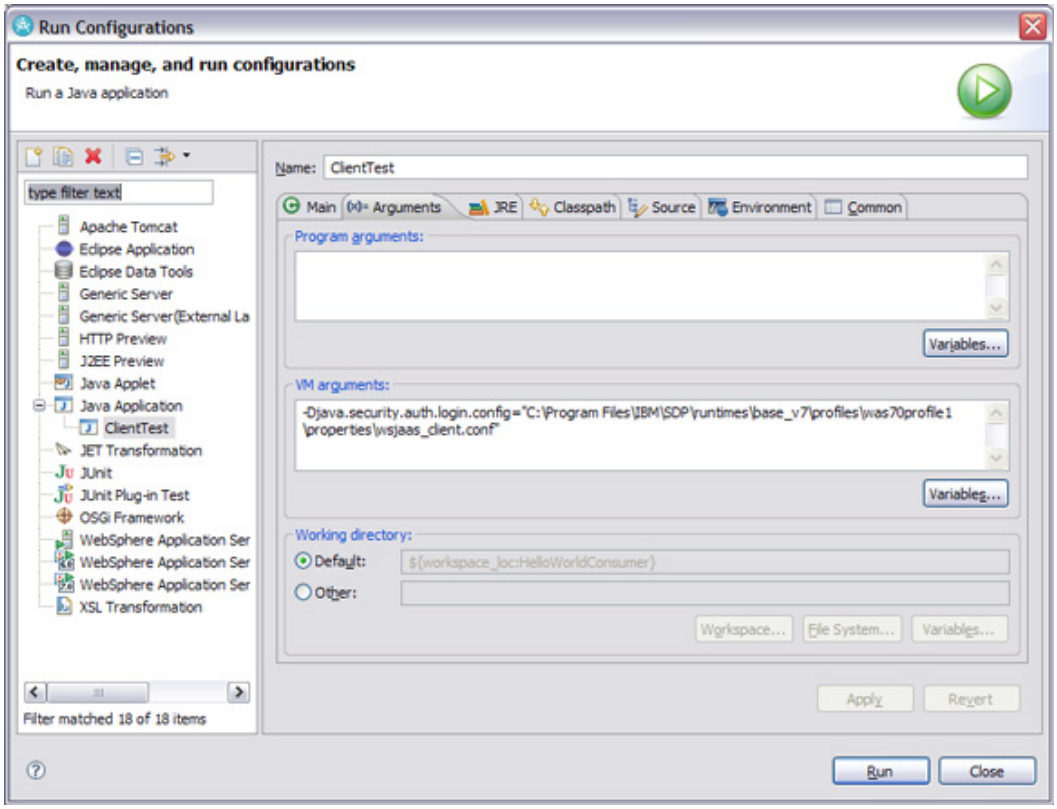
1. From Application Developer, right-click **ClientTest.java** and select **Run**

As => Run Configurations.

2. As shown in Figure 19, since the consumer needs to use Java Authentication and Authorization Service (JAAS) in order to pass in the Username credentials, specify the following for **VM arguments** to point to the JAAS login configuration file:

```
-Djava.security.auth.login.config="C:\Program
Files\IBM\SDP\runtimes\base_v7\profiles\was70profile1\properties
\wsjaas_client.conf"
```

Figure 19. Set JAAS arguments for ClientTest



3. Click **Run**. You should see client results as shown in Figure 20 and server-side results as shown in Figure 21.

Figure 20. JAX-WS consumer results

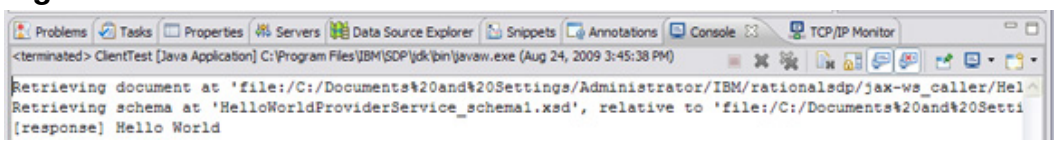
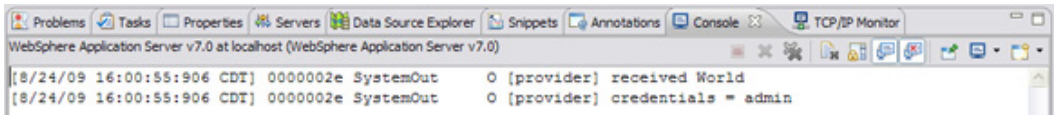


Figure 21. JAX-WS provider results



Summary

Many Web services require authorization in addition to authentication, integrity, and confidentiality. In this article, you've learned how to configure WebSphere Application Server V7 to choose a security token that is part of the SOAP header as the JEE security principal. Since this configuration is done at the binding level for the policy set, each Web service port could have a different configuration, if desired, or you can specify the configuration at the service level as we did in this article. Once this configuration has been set, the JEE authorization APIs are available to developers so that authorization decisions can be made. For EJB-based Web services, the configured JEE principal can be used for JEE role-based authorization checking using annotations or deployment descriptors.

Acknowledgement

The authors would like to thank Bill Dodd for his thorough review of this article.

Downloads

Description	Name	Size	Download method
Sample project interchange	jax-ws-caller_PI.zip	21KB	HTTP
Sample policy set	My UNT.zip	1KB	HTTP

Information about download methods

Resources

Learn

- Authorization concepts and solutions for J2EE applications (developerWorks, 2006)
- WebSphere Application Server Information Center: Role-based authorization
- Redbook: IBM WebSphere Application Server V7.0 Web Services Guide
- Redbook: Web Services Feature Pack for WebSphere Application Server V6.1
- Redbook: WebSphere Application Server V7.0 Security Guide
- Redbook: Rational Application Developer V7.5 Programming Guide
- WebSphere Application Server V7 Information Center
- developerWorks WebSphere Application Server zone
- developerWorks Web services and SOA zone
- WebSphere Application Server forum

Get products and technologies

- Download Rational Application Developer V7 trial
- Download IBM SOA Sandbox for reuse

About the author

Henry Chung
Short bio