WA1791 Designing and Developing Secure Web Services

Student Labs

Web Age Solutions Inc.

Table of Contents

Lab 1 - WebSphere Workspace Configuration

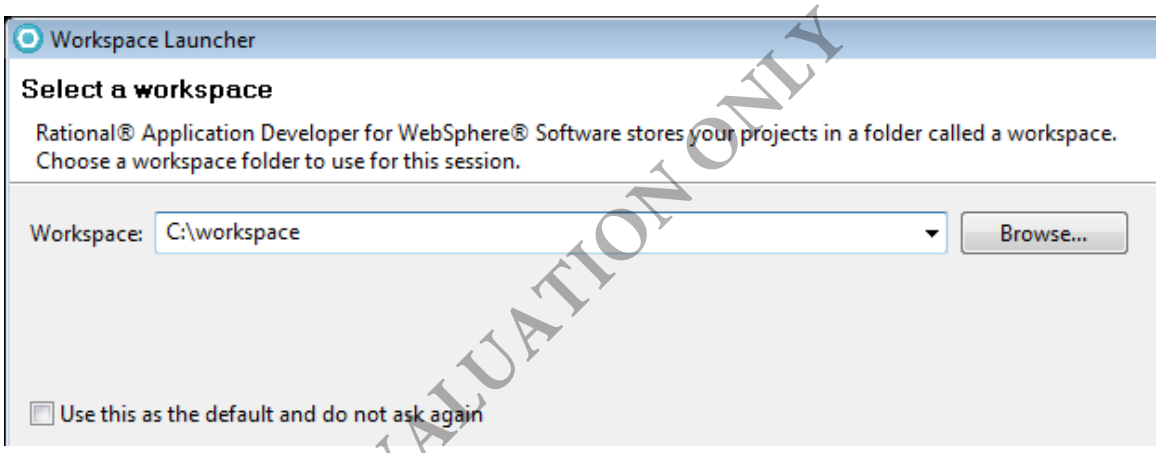
In this lab you will configure the RAD workspace to work with a WebSphere test environment server. This will include creating a WebSphere "profile" which will have the server definition. It is sometimes useful to have different profile configurations for testing purposes so this process is good to be familiar with.

Part 1 - Create Profile

The WebSphere Application Server software is installed along with RAD. In this section you will run a standard WebSphere tool, the Profile Management Tool, to create a profile.

__1. From the Windows Start menu select **Start → Programs → IBM Software Delivery Platform → IBM Rational Application Developer 8.0 → Rational Application Developer**.

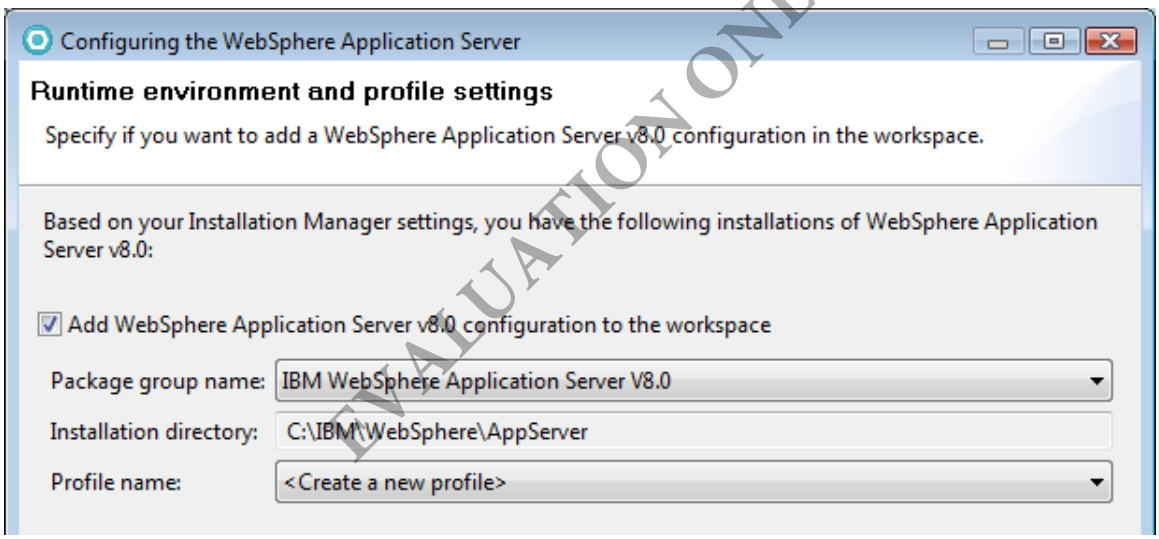
__2. In the workspace window, change the **Workspace** to **C:\workspace**



__3. Click **OK**.

RAD 8.0.4 will start.

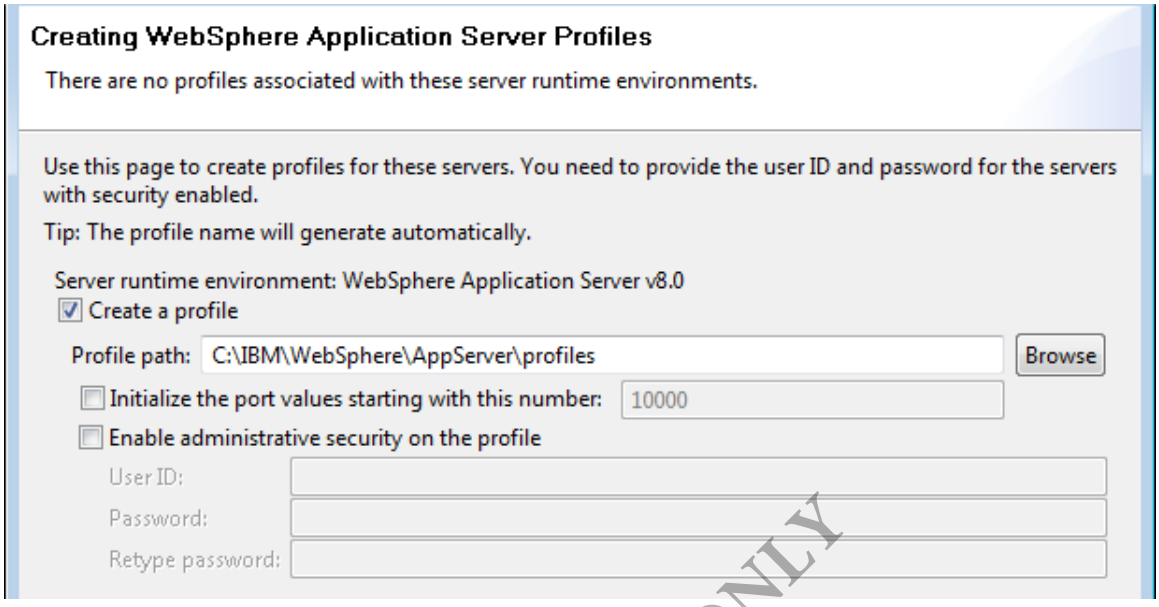
__4. In the *Runtime environment and profile settings* dialog that appears check that the option to '**<Create a new profile>**' is available and click the **Next** button.



Note: If you **DO NOT** get this prompt it is possible that a profile was created for this workspace already. Since it is always best for the user who will be using the workspace to create the profile restart RAD and use a different workspace with RAD. Inform your instructor of this to see if it is a common problem in the class.

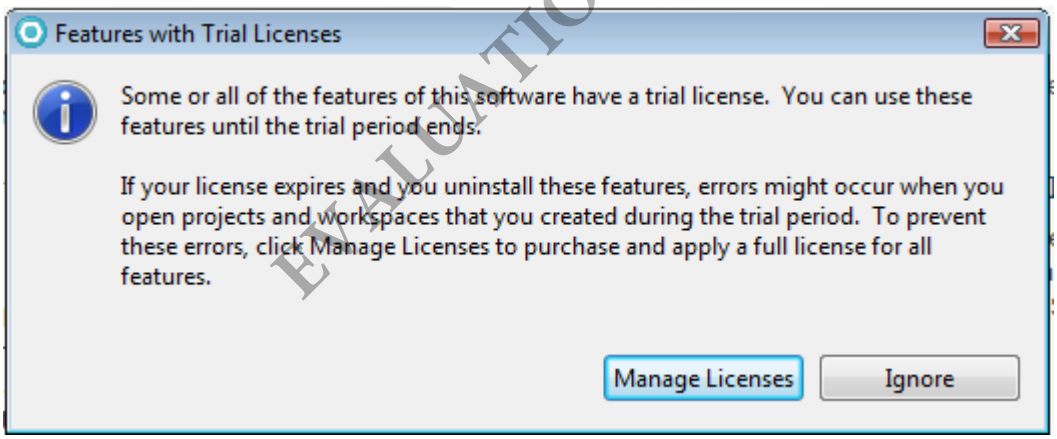
If there is an existing profile listed in the dialog a previous profile may already exist. Let your instructor know about this because although this profile can likely be used with your workspace, if it has applications deployed to it from another workspace, there may be errors when starting the server. It is also possible to run the 'Profile Management Tool' to create a new profile but this will be complex and should be guided by the instructor.

__5. **Uncheck** the '**enable administrative security on the profile**' option as shown below and click the **Finish** button.



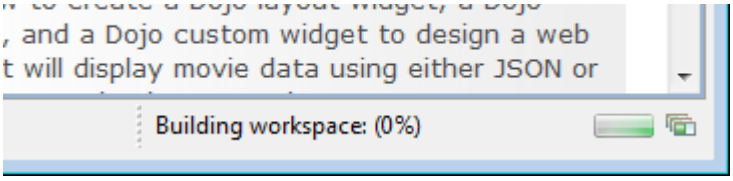
The **Features with Trial Licenses** dialog will open.

__6. Click **Ignore**.



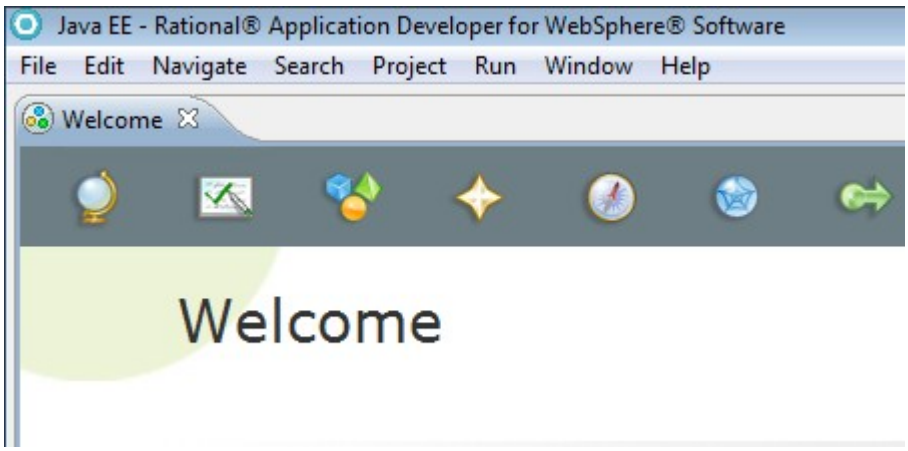
__7. If prompted about help content select **Work without Help**.

__8. Wait until the messages in the lower right corner of RAD disappear to know the work of creating the server is complete.



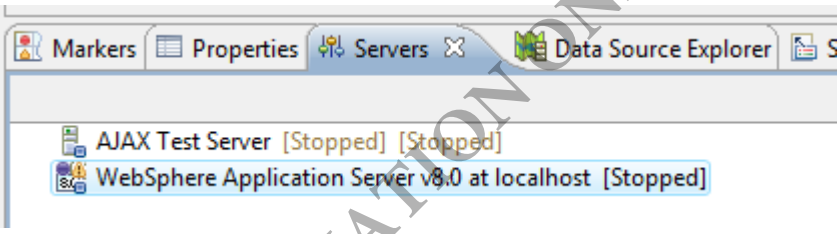
__9. If prompted by Windows firewall security hit the **Cancel** button.

__10. Close the **Welcome** page if it appears.



__11. Select the **Servers** view on the bottom of the *Java EE* perspective.

__12. Check that a WebSphere Application Server v8.0 server is shown.



__13. Open the following file:

```
C:\IBM\WebSphere\AppServer\logs\manageprofiles\AppSrv1_create.log
```

__14. Look at the end of the file for the message:

```
<record>
    <date>2012-07-08T04:28:29</date>
    <millis>1341779309892</millis>
    <sequence>4676</sequence>
    <logger>com.ibm.wsspi.profile.WSProfileCLI</logger>
    <level>INFO</level>
    <class>com.ibm.wsspi.profile.WSProfileCLI</class>
    <method>invokeWSProfile</method>
    <thread>0</thread>
    <message>Returning with return code: INSTCONFSUCCESS</message>
</record>
</log>
```

If you see the message INSTCONFSUCCESS, it means that the WebSphere Application Server profile was created correctly.

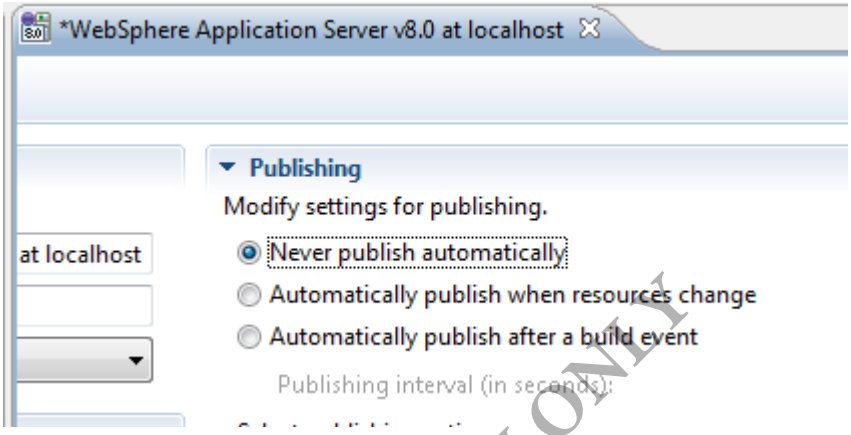
__15. Close the log file.

__16. Switch back to RAD and the **Servers** view.

__17. Make sure the server is stopped.

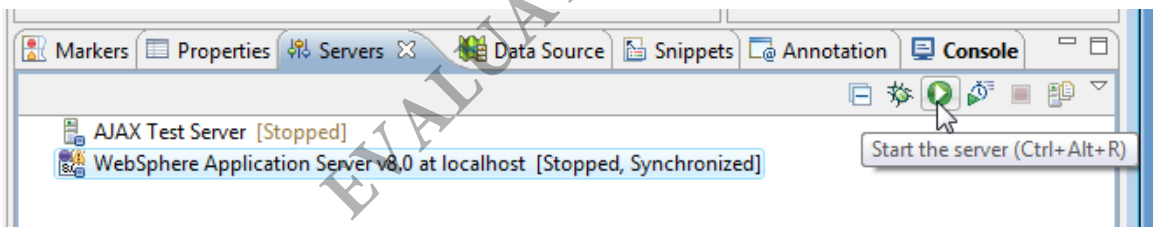
__18. Double click on the server to open the server properties.

__19. On the right, expand **Publishing** and select the '**Never publish automatically**' option.



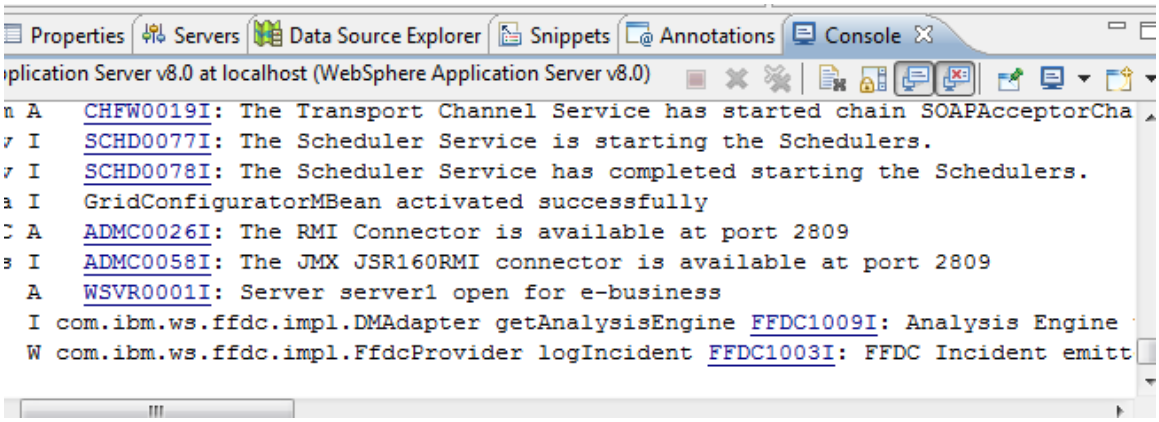
__20. Save and close the server properties.

__21. Select the **WebSphere Application Server v8.0** server that was created in the *Servers* view and click the **Start the server** button on the right.

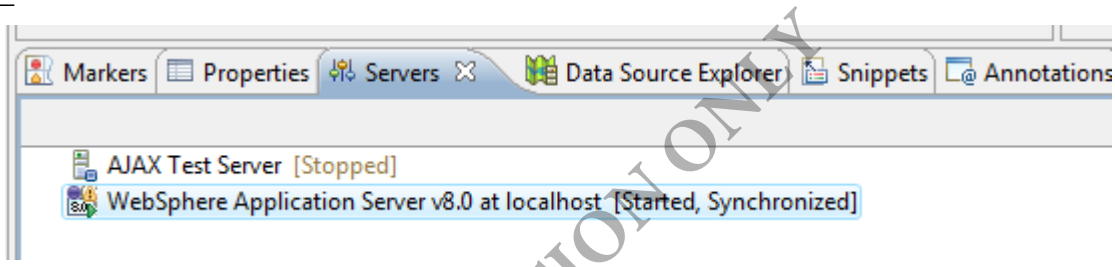


__22. From the menu, select **Window > Show View > Console**.

__23. Check the *Console* view as the server starts to look for errors.

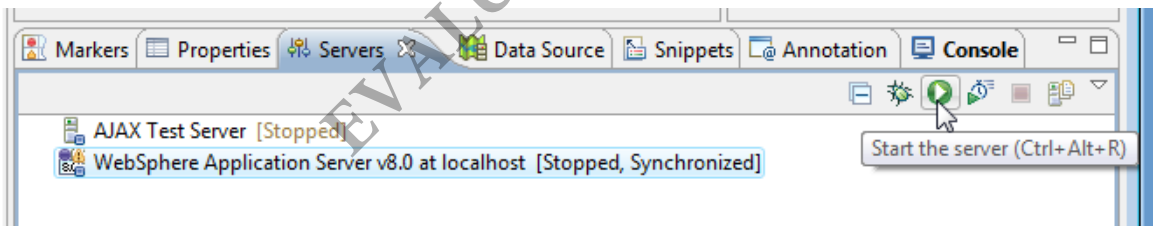


__24. Switch back to the *Servers* view and check that the server was started.



__25. Right click the server and select **Stop**.

__26. Check that the server was Stopped.



Part 2 - Review

You have now configured a test server to use with the RAD workspace.

Lab 2 - Getting Started

Throughout this class, we will apply various WS-Security protections to a web service provider and consumer. In this lab, we will install and test these applications before any security is enabled.

Tip: Always function test a provider and consumer before security is enabled. This will help you debug problems when security related problems begin to appear.

The business logic for these applications is very simple. The provider is called BillingManagerService. It has only one operation called addAccount. A consumer calls this operation to create a new customer account.

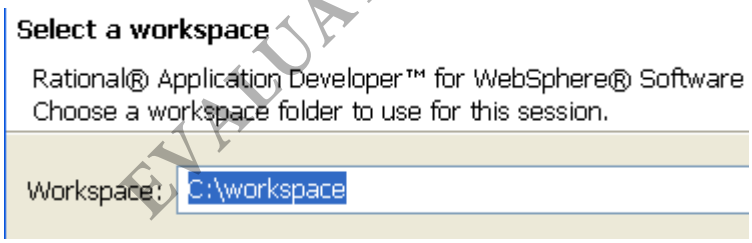
The consumer is a web based application. When a new account form is submitted, it calls the addAccount operation of the web service.

Part 1 - Review Code

We will briefly review the code of the provider and consumer. They have been developed using the JAX-WS API. There is nothing special about them. In other words, there is nothing done to them specific to WS-Security. All security settings will later be applied through policies.

__1. Select **Start > All Programs > IBM Software Delivery Platform > IBM Rational Application Developer 7.5 > IBM Rational Application Developer**.

__2. Enter **C:\workspace** as the workspace folder.



__3. Click **OK**.

__4. Close the **Welcome** screen.

__5. Click **Ignore** if the **Warning: Help content not installed dialog** opens.

__6. Close any dialog that may open related with the Trial version.

__7. From the menu select **File > Import**.

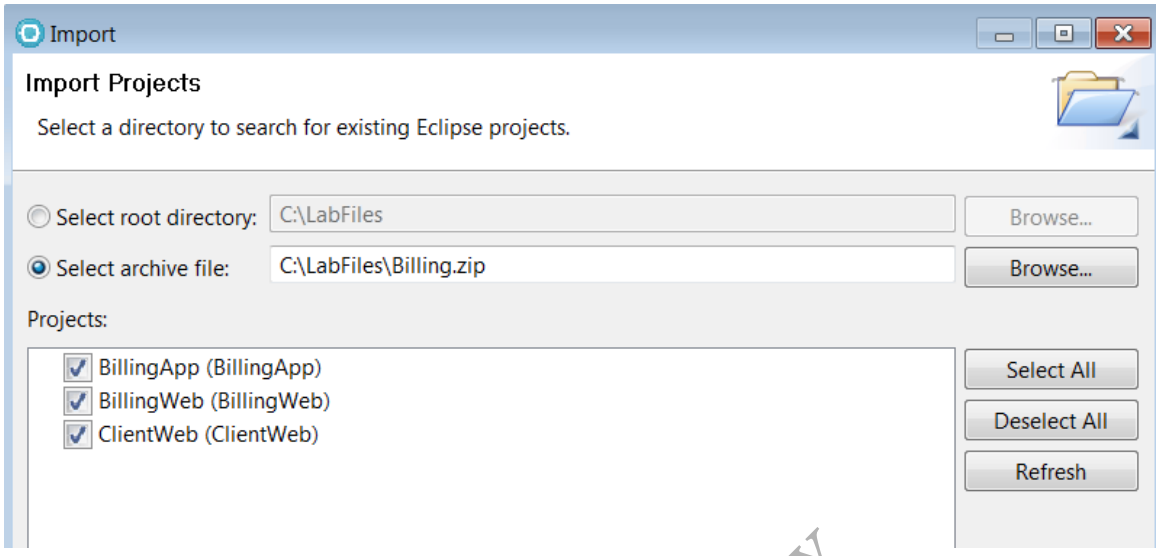
__8. Expand **General** and select **Existing Projects into Workspace**.

__9. Click **Next**.

__10. Click the second **Browse** button next to *Select archive file*.

__11. Select the ZIP file **C:\LabFiles\Billing.zip** and click **Open**.

__12. Click **Select All** to select all projects.

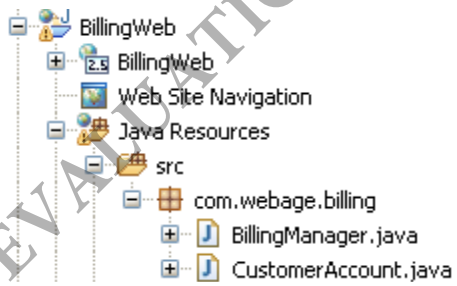


__13. Click **Finish** to import the projects.

First, we will review the service provider.

__14. Expand the **BillingWeb** project.

__15. Then expand **Java Resources > src > com.webage.billing**.



__16. Double click **BillingManager.java** to open the provider implementation class.

__17. Notice that the @WebService annotation is used for the class.

```
@WebService
public class BillingManager {
```

__18. The class has only one public method called addAccount. Various JAX-WS annotations are used with that method to customize the XML schema for the request and response.

```
@WebMethod
@WebResult(name="status")
public String addAccount(
            @WebParam(name="account")
            CustomerAccount account) {
```

This is pretty basic JAX-WS API. If you have any questions about them ask the instructor.

__19. Note that the addAccount operation simply prints out a few lines in the log file and finally returns "OK".

__20. Close the editor.

Now, we will review the consumer.

__21. Expand **ClientWeb** project.

__22. Expand **WebContent**.

__23. Right click **index.jsp** and select **Open With > JSP Editor**.

__24. Observe the following:

- The JSP file has a form that is submitted to index.jsp
- If the request method is POST, the BillingManagerPortProxy class is used to invoke the addAccount operation.

__25. Close the editor.

Part 2 - Deploy the Applications

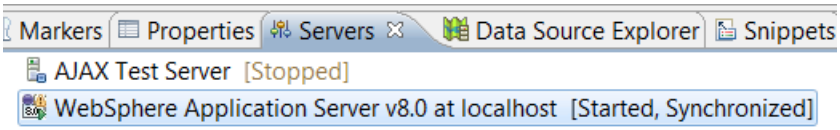
Normally, as a developer, you would deploy the projects to the server using RAD. But, in the following labs, we will do things like the administrators. That means, we will export an EAR file and deploy the EAR using WebSphere admin console. This will give us a feel for how policies are actually configured and attached to service providers and consumers in real life. That approach will also drive home the point that no code change is required to enable security.

__1. Open the **Servers** view.

__2. Select **WebSphere Application Server v8.0 at localhost**; for future references we will call it 'the Server'.

__3. Right click on it and select **Start**.

__4. Wait until the status of the server becomes **Started**.



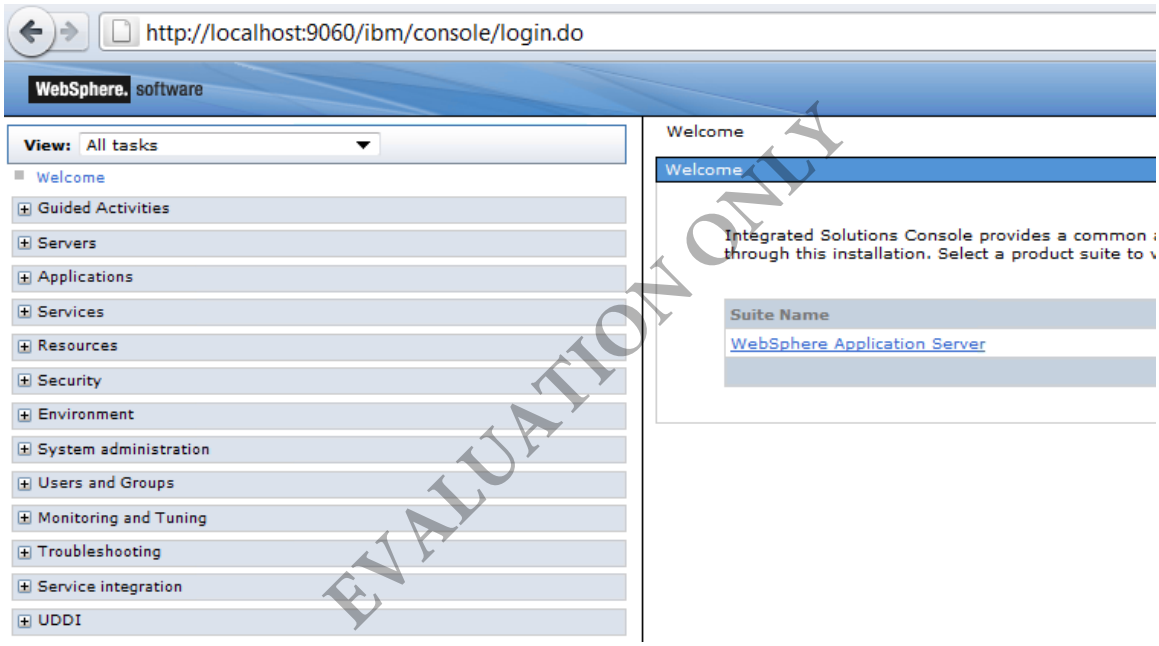
__5. Launch a browser window like IE or FireFox.

__6. Enter the URL:

```
http://localhost:9060/ibm/console/
```

__7. Enter anything as user ID (such as www) and log in.

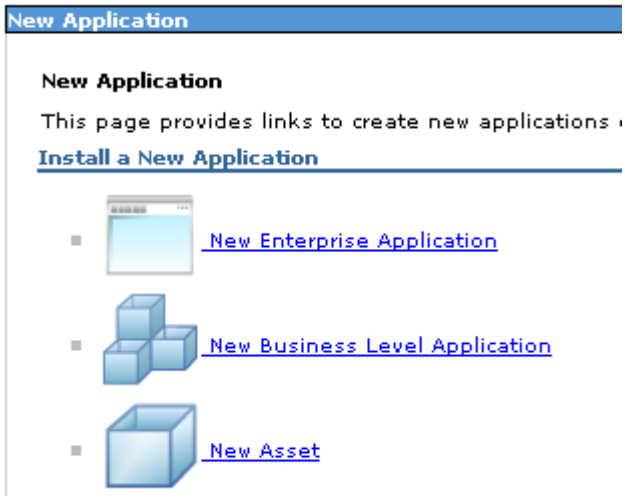
The admin console will open.



__8. Expand **Applications**.

__9. Click **New Application**.

__10. Click **New Enterprise Application**.

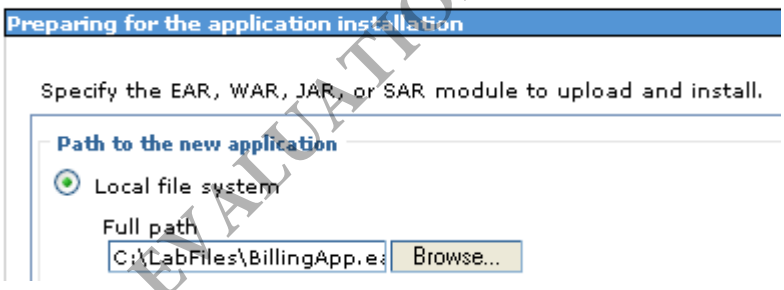


__11. Make sure that **Local file system** is selected.

__12. Click **Browse**.

__13. Select **C:\LabFiles\BillingApp.ear** and click **Open**.

__14. Click **Next**.



__15. Leave the default in the **How do you want to install the application?** page and click **Next**.

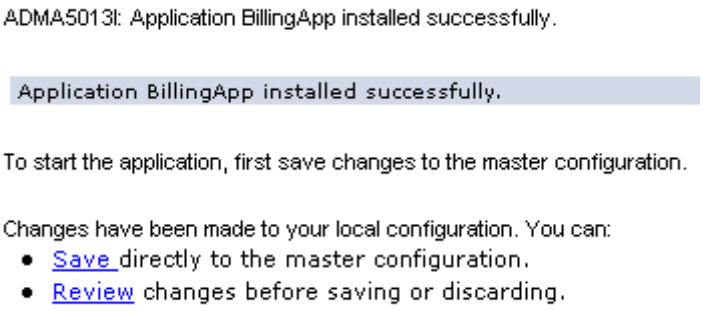
__16. In the **Step1: Select installation options** page just click **Next**.

__17. In the **Step 2: Map modules to servers** page, again just click **Next**.

__18. In the **Step 3: Metadata for modules** page, again just click **Next**.

__19. Click **Finish**.

__20. The application will be installed, wait until you see the following message:



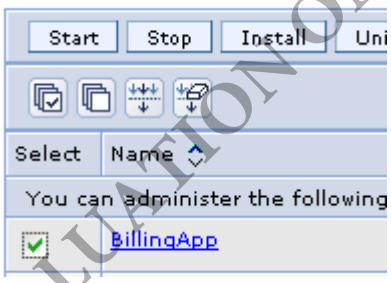
__21. Click **Save**.

Part 3 - Test the Application

__1. On the left hand side, expand **Applications > Application Types**.

__2. Click **WebSphere enterprise applications**.

__3. Select the check box next to **BillingApp**.



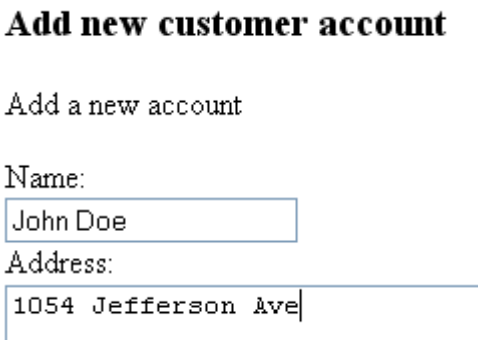
__4. Click the **Start** button.

__5. Launch a new web browser window or tab.

__6. Enter the URL:

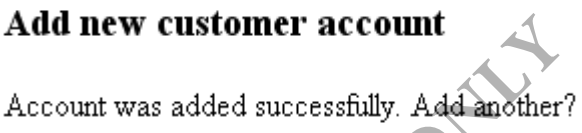
```
http://localhost:9080/ClientWeb/index.jsp
```

__7. The page will be displayed. Add some input as shown below.



__8. Click **Add**.

__9. The JSP will call the web service. Make sure that the page shows the success message:



__10. Back in RAD, the **Console** view will show the log output from the web service.

```
3013 SystemOut     O Retrieving schema at 'BillingMa
3013 WSChannelFram A   CHFW0019I: The Transport Char
3017 BillingManage I   Adding new account
3017 BillingManage I   Customer name: John Doe
3017 BillingManage I   Address: 1054 Jefferson Ave
```


Part 4 - Enable SOAP Monitoring

We will use the TCP monitor tool to inspect SOAP messages.

__1. In the **Servers** view, right click the server and select **Monitoring > Properties**.

__2. Click **Add**.

__3. Select the port **9080**

The **Monitor port** will be set to 9081 by default. That means, all HTTP requests submitted to localhost:9801 will be forwarded to 9080.

__4. Click **OK**.

__5. Select the newly added monitor port and click **Start**.

__6. Click **OK**.

By default, the consumer submits SOAP request to port 9080. We need to change it to 9081.

__7. Open a file browser and navigate to the folder:

```
C:\IBM\WebSphere\AppServer\profiles\AppSrv1\installedApps\hostnameNode01Cell\Bi
llingApp.ear\ClientWeb.war\WEB-INF\wsdl
```

Note: You need to use your hostname as shown in bold below. In this example:

```
C:\IBM\WebSphere\AppServer\profiles\AppSrv1\installedApps\WA2215Node01C
ell\BillingApp.ear\ClientWeb.war\WEB-INF\wsdl
```

__8. Open the file **BillingManagerService.wsdl** using Wordpad.

__9. Scroll to the end of the file and then change the SOAP endpoint to port 9081.

```
<soap:address
location="http://localhost:9081/BillingWeb/BillingManagerService"/>
```

__10. Save and close the editor.

We will now restart the application and test the consumer.

__11. Back in admin console web page, select the checkbox for **BillingApp**.

__12. Click **Stop**.

__13. Select again the checkbox for **BillingApp**.

__14. Click **Start**.

__15. Go back to the consumer web page (http://localhost:9080/ClientWeb/index.jsp).

__16. Enter some input data and click **Add**.

__17. Make sure that the success message is shown.

We will now inspect the SOAP messages.

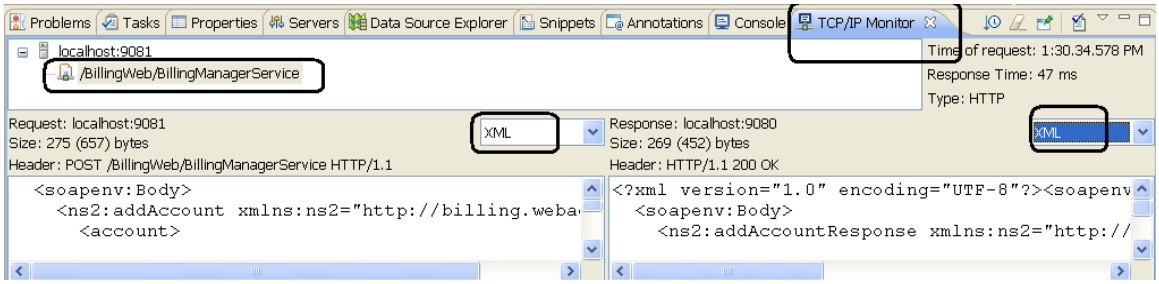
__18. In RAD, select **Window > Show View > Other**.

__19. Expand **Debug** and select **TCP/IP Monitor**.

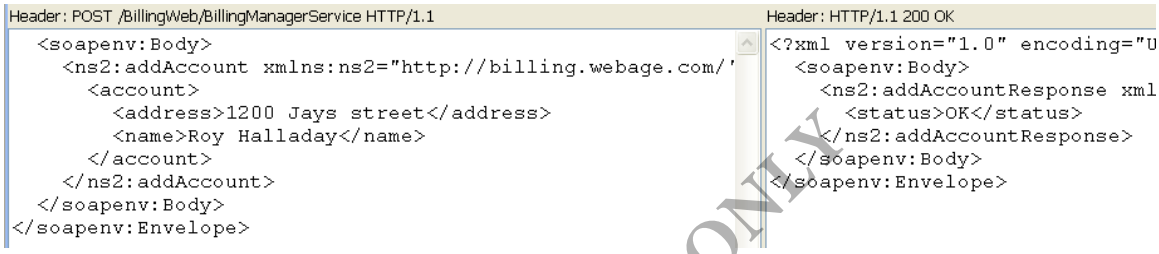
__20. Click **OK**.

__21. Maximize the **TCP/IP Monitor** view.

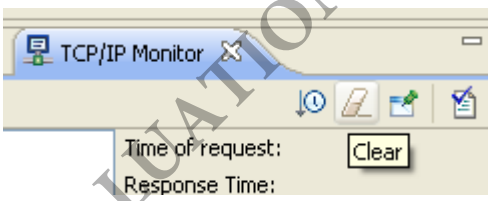
__22. You should see a request already captured. Select it as shown below. Also, set the display type to XML.



__23. Study the request and response messages. They are in plain text since we have not enabled any message security.



__24. Click the eraser toolbar in TCP monitor to remove all captured messages.



__25. Restore the size of the **TCP/IP Monitor** view.

__26. Close all web browsers.

Part 5 - Review

In this lab, we installed a basic web service provider and consumer. They have been developed using the JAX-WS API.

We tested the applications to make sure that they are functionally correct.

We have enabled TCP monitoring so that we can inspect the SOAP messages.

In the subsequent labs, we will enable various message protection features.

Lab 3 - Message Integrity using WS-Security

In this lab, we will implement message integrity for the BillingManagerService provider and consumer. But, before that we will review the basics of WS-Security.

The BillingManagerService web service so far has been operating without any form of security. Our clients have been sending SOAP messages to the web service in clear plain text, with no regard for security at all.

This is clearly not practical in many production environments. For a web service, the concept of security has three concerns:

Integrity: Integrity ensures that a message came from a unique party, and that the message has not been tampered with in transmission. This is achieved by *signing* the outgoing SOAP message. In this lab exercise, we will implement message integrity.

Authority: Ensures that an invocation of a web service is only allowed for authenticated clients. This is achieved by adding a username/password token to the message.

Confidentiality: Ensures that a message being transmitted can only be understood by the receiving party. This is achieved by encrypting the outgoing SOAP message.

Ideally, we would like to add some combination of these 3 factors into our web service layer. Invoking a service should be done with integrity, with authority and with confidentiality.

But how can we do this? The hard way would be to add code to our SOAP layer. Remember that SOAP (XML) is the underlying communication mechanism for a client and service. For confidentiality, we could go about adding code to our service to return encrypted responses; for integrity, we could add code our client to attach a digital signature. This would involve an extremely long and complex series of coding additions to our client.

Fortunately for us, the JAX-WS spec can help us avoid having to do just that. JAX-WS, instead makes use of *policies*, which will provide all the security layers that we need.

Security in web services is handled by using the concept of *policy set*. A policy set is a set of *policies*; policies are rules (defined by a specification) specifying what features should be enabled. For example, a policy can be created stating that *integrity* should be enabled; or *integrity* and *authority*; or even integrity and authority and encryption. (Multiple policies can be combined into a single policy set). These policies are defined by the WS-Security specification, and any JAX-WS compliant run-time must support these policies.

Once a policy has been created (in the form of an XML file), it is attached to a service, using vendor tools. No code changes have to be made. Now, when a client attempts to invoke the service, the container (WebSphere) will insist that the client encode the request message appropriately (e.g. encrypting the message, attach a signature, etc). If the client does not have the appropriate security measures in place, the service will reject the client.

So, we know how get the service to require security; but how do we get the client to attach the required security? Again, the answer is the use of a policy. A policy can also be attached to a client. The client stub will read the attached policy and at invocation time, the client stub will generate the required SOAP – encrypting, attaching signatures, etc- as necessary.

This means our client code does not have to change; we just have to make sure we attach the policy set to the client stub, and the client stub will handle all the rest.

In this lab, we will examine a simple policy set to enable security; we will then attach it to a service (and client) and invoke it.

Part 1 - Understanding Policy Set

A policy set is a ZIP file consisting of a file called **policySet.xml** as well as one or more **policy.xml** files, each in its own sub-directory. Each **policy.xml** file contains a list of features to be enabled. The **policySet.xml** file then contains a list of which **policy.xml** files to use.

Once a policy set ZIP file has been created, it needs to be *imported* into the run-time (i.e. RAD and WebSphere). When imported, the run-time will examine set the policy set's **policySet.xml** and, in turn, read each **policy.xml** that is referenced.

Out of the box, several policy sets are available. For example, WebSphere and RAD come bundled with a policy set called "**WS-ReliableMessaging"** - which enables reliable messaging.

In this lab, however, we will use our own policy set. We will now briefly inspect a policy set ZIP file. We will inspect the policies in more details from within WebSphere adminconsole.

__1. Using a Windows file explorer, navigate to **C:\LabFiles**. Notice that it contains a file called **IntegrityOnly.zip**. This is the policy set has a policy that enables message integrity.

__2. Open the ZIP file using Winzip or any other archiving program.

__3. In the **PolicySets\IntegrityOnly** folder, locate the **policySet.xml** file.

__4. Open the **policySet.xml** file. Note that the set includes a single policy as follows:

```
<ps:PolicyType type="WSSecurity" provides="" enabled="true">
</ps:PolicyType>
```

__5. Close the file.

__6. Open the **PolicyTypes** sub-folder. In here, there is a sub-directory called **WSSecurity –** which is exactly what the **PolicyType** element in **policyTypes.xml** referred to. Basically, the run-time will read the **policyTypes.xml** file and then search for an appropriate sub-directory with the same name.

But what goes into the policy type sub-directory?

__7. Open the **WSSecurity** folder.

It contains a single file – **policy.xml.** This file specifies what features to activate.

__8. Open **policy.xml** with a text editor.

Ouch. This is a fairly complex XML file. For now, focus on these elements:

```
    <wsp:Policy wsu:Id="request:app_signparts">
        <sp:SignedParts>
            <sp:Body/>
...
        </sp:SignedParts>
```

This enables signing of the SOAP <Body> element.

__9. Close the file and close the ZIP file.

Part 2 - Import the Policy Set

Before a policy set can be attached to a service provider or consumer, we must import it in the WebSphere profile. We will do so now.

__1. In RAD, locate the **Servers** view.

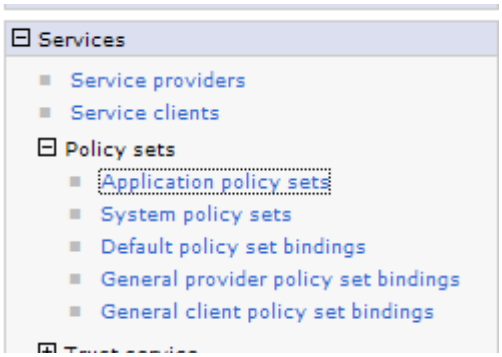
__2. Start the server if it is not running.

__3. When the server is started, right-click on the server (in the *Servers* view) and select **Administration | Run administrative console**.

The admin console will open inside RAD.

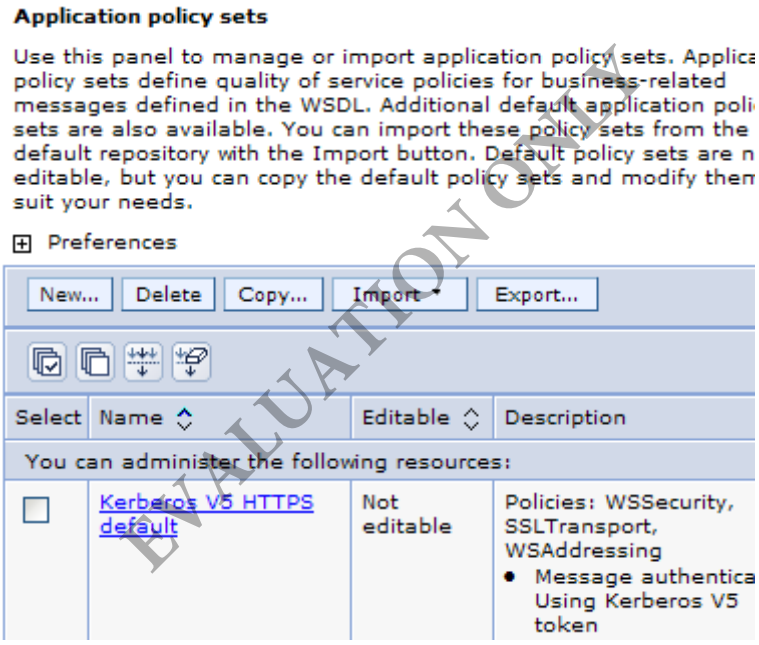
This is the administrative front end to WebSphere that you saw in the previous Lab, and is where a WebSphere administrator would perform the majority of administration tasks.

__4. In the left pane, expand **Services | Policy Sets**.



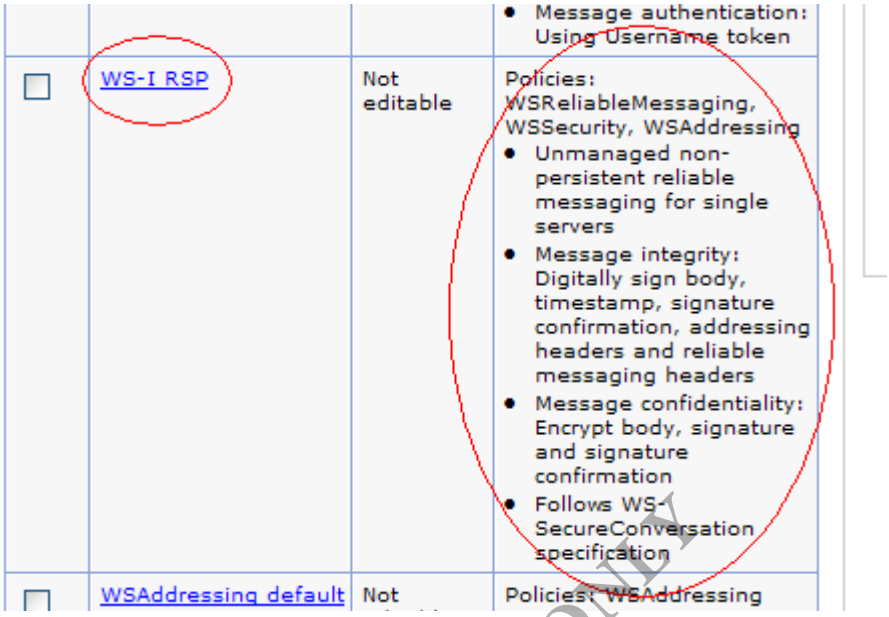
__5. Click **Application policy sets**.

The right pane will display the *Application policy sets* screen.



This lists all the policy sets that are currently deployed to WebSphere. As mentioned above, there are a few "out of the box" policy sets available.

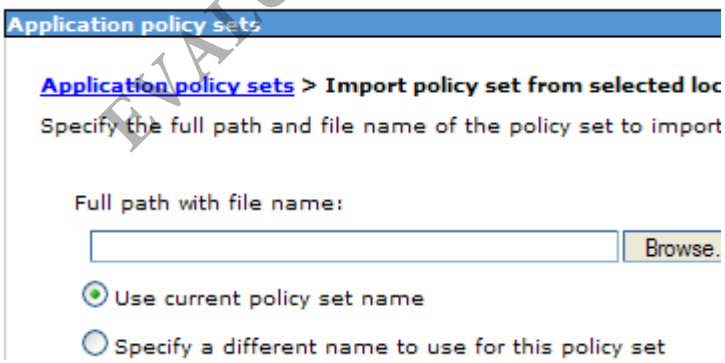
__6. Scroll down the list to examine some of them and what they offer. For example, examine the **WS-I RSP** profile, and look at what it offers.



Instead of using one of these existing profiles, however, we will use the one we examined previously. To do so, we need to import it here.

__7. Scroll up. Click the **Import** button, and select **From Selected Location...** from the drop-down.

The screen will change to allow you to specify a location for the policy set.

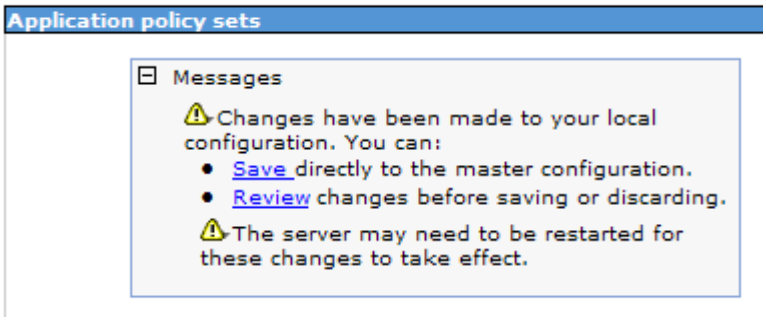


__8. Click the **Browse...** button.

__9. Select **C:\LabFiles\IntegrityOnly.zip** and click **Open**.

__10. Click **OK**.

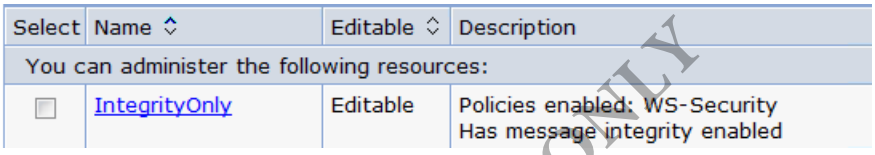
A *Messages* pane will appear.



__11. Click the **Save** link.

The policy set will be imported.

__12. In the list of policies, scroll down. You should see the newly imported policy set.



Part 3 - Attach Policy Set

To enable message integrity, we need to attach the policy set to the provider. And since the consumer is also running in WebSphere, we need to attach the policy set to it as well. If the consumer was running in a different platform, say .NET, you will have to use vendor specific ways to enable integrity. If you do not enable integrity at the client level, the server will reject a SOAP request that does not contain a digital signature of the message.

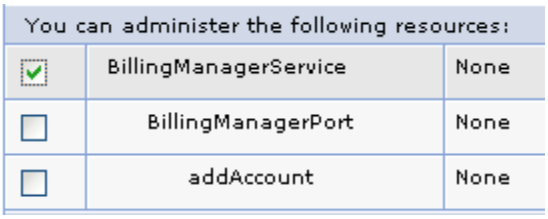
First, we will attach the policy set to the provider.

Note: A policy set can be attached to a provider or consumer using either RAD or WebSphere adminconsole. The former is perhaps easier during development. Here, we will do things like an administrator would in a production environment. That means, we will attach policy sets using the adminconsole.

__1. On the left hand side of admin console, expand **Services** and click **Service providers**.

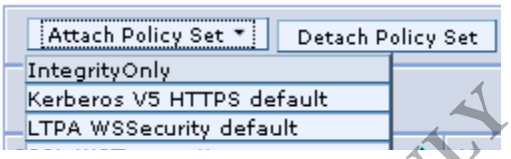
__2. Click **BillingManagerService**.

__3. Select the checkbox for **BillingManagerService**.



This will select the entire service so that we can attach the policy set to it. Alternatively, you can select a specific operation, such as addAccount, and attach a policy set to it. That way, different operations can have different policies. In this lab, we will keep things simple and attach the policy set for the whole service.

__4. Click the **Attach Policy Set** button and click **IntegrityOnly**.



Now, we need to attach a policy set binding to the provider.

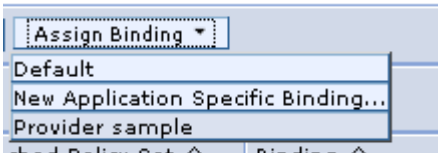
What is a **policy set binding**? A policy simply states what communication behavior should be enabled. For example, integrity and encryption. Various encryption and digest algorithms are also specified in the policy. A policy *does not* specify machine or installation specific details. For example, it does not describe the location of the private key and the password to open the key store file. Policy set binding is used to describe those items.

A policy set is generic and not specific to any business or installation of application server. For example, the same policy set ZIP file can be copied from development machine to production without any modification. A policy set binding ZIP file, on the other hand, contains installation specific details.

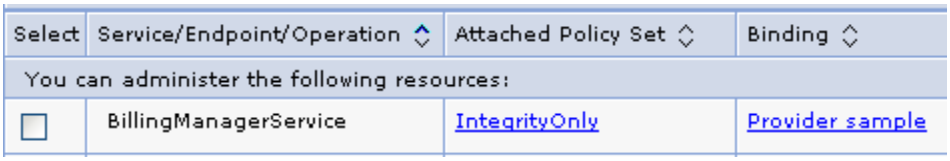
In this and the next few labs, we will use a predefined policy set binding. This binding uses pre-generated keys for the provider and consumer. This is good enough for development. In a production environment, you must generate your own keys. Preferably a key should be signed by a well known certificate authority. We will learn about key management and custom binding design in a latter lab.

__5. Select the checkbox for **BillingManagerService** again.

__6. Click **Assign Binding > Provider sample**.



__7. Verify that the correct policy set and binding is displayed for the service.



__8. Click the **Save** link at the top of the page to save changes.

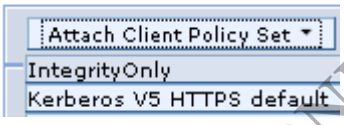
Now, we will configure the consumer.

__9. On the left hand side, click **Service clients** under **Services**.

__10. Click **BillingManagerService**.

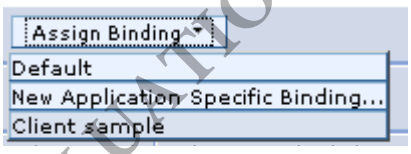
__11. Select the checkbox for **BillingManagerService**.

__12. Click **Attach Client Policy Set > IntegrityOnly**.



__13. Select the checkbox for **BillingManagerService** again.

__14. Click **Assign Binding > Client sample**.



__15. Verify that the correct policy set and binding is displayed for the consumer.

__16. Click the **Save** link at the top of the page to save changes.

Now, integrity should enabled. Before we can test things, we need to restart the server.
Note: In WebSphere, server must be restarted after policy set and binding attachments have been modified for a service provider or client.

__17. Log out of admin console by clicking the **Logout** link at the top right corner.

__18. Close the browser.

__19. Select the server in **Servers** view.

__20. Press Control+Alt+R to restart the server or right click the server and select **Restart**.

__21. Wait for the server status to change to **Started**.

Part 4 - Test Message Integrity

__1. First make sure that the TCP monitoring proxy service is still running. To do that, right click the server and select **Monitoring > Properties**. Make sure the status is **Started**. Otherwise, start it.

__2. Click **OK** to close the monitor settings dialog.

__3. Open a browser window and enter the URL for the consumer application:

```
http://localhost:9080/ClientWeb/index.jsp
```

__4. Enter some input value. Then click **Add**.

__5. Make sure that the success message is displayed.

__6. Back in RAD, open the **TCP/IP Monitor** view and maximize it.

__7. Change the display mode from **Byte** to **XML** for both request and response.

__8. For the request message, observe the SOAP <Body> element of the **Request** is still in plain text. This is expected since we have not enabled encryption.

__9. In the header of the **Request**, locate the **<ds:SignatureValue>** element.

```
<ds:SignatureValue>r43pj30kuKqKy ... Mpl0m994z8Rk=</ds:SignatureValue>
```

This contains the digital signature of the message.

The digest of the message is also available in the <DigestValue> element as shown below.

```
<ds:DigestValue>EVFKYcBANxvWQavJm3spuddWNrk=</ds:DigestValue>
```

But, that is rather redundant. The signature is the digest encrypted by the consumer's private key.

The consumer sends its public key in the form of its certificate. You can see that in the BinarySecurityToken element.

A BinarySecurityToken can carry any token that uniquely identifies the consumer. The ValueType attribute clearly describes the nature of the token. In our case, it is ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#**X509v3**". This value type indicates that the token is a X509 certificate.

Why does the client send its certificate? The provider uses that to decrypt the signature and obtain the digest. If encryption is enabled, the provider uses the public key from the certificate to encrypt the SOAP response message.

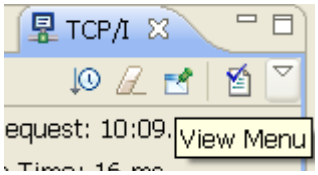
__10. Similarly, locate and observe the SignatureValue, DigestValue and BinarySecurityToken elements in the SOAP **response** message.

Part 5 - Test for Integrity Failure

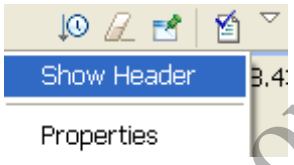
Now, we will mount a man in the middle attack. We will alter the request message and resend it. The TCP/IP Monitor view allows us to modify an existing request and resend it.

First, we need to view the HTTP header of the existing request. If we do not do that, headers will be omitted from the modified request. Which appears to be a bug in RAD.

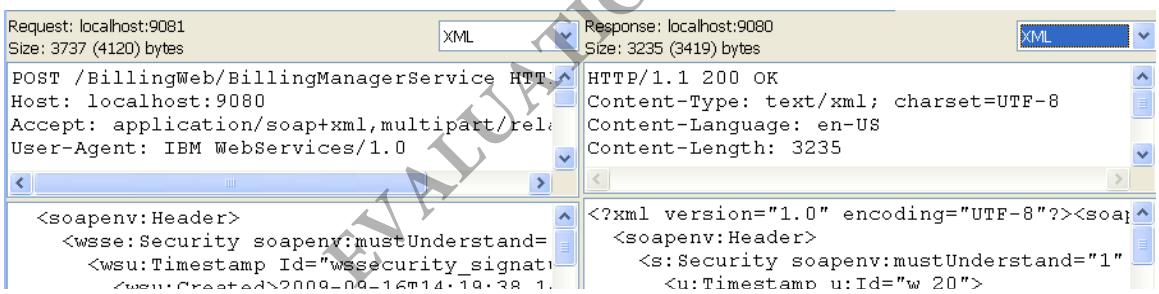
__1. In the **TCP/IP Monitor** view, move your mouse over the low-arrow at the top-right of the menu bar, it will display **View Menu**.



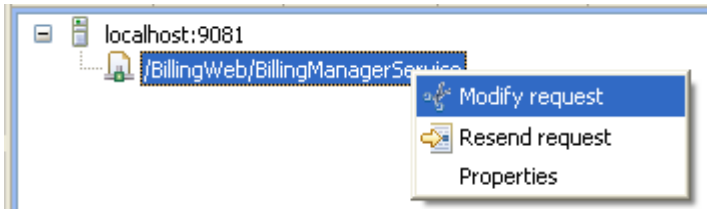
__2. Click the **View Menu** button and select **Show Header**.



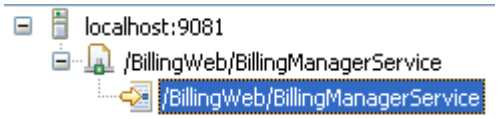
__3. The **TCP/IP Monitor** view now will show the header for the Request and Response.



__4. In the top pane of the **TCP/IP Monitor** view, right click the request and select **Modify request**.



The tool will create a modified request.



Note: The request can be edited in Byte mode only. If you change the display mode to, say, XML, the editor will become read only.

__5. In the **Request** editor, scroll carefully to the right and locate the <**address**> element in the bottom line.

__6. Replace some of the characters in the body of the <address> element. For example, change 1 to 2. Or, change an upper case letter to lower case. **Important:** Make sure that the total number of characters remain the same.

__7. Right click the modified request and select **Send Modified Request**.

__8. The **Console** view will show an exception from the service.

__9. Back in the TCP/IP Monitor view, for the **Response** pane, change the display mode to **XML**.

__10. Notice that a fault has been returned by the server:

__11. Clear all requests in TCP/IP Monitor by clicking the eraser toolbar button.

__12. Hide the Header since you don't need it anymore, remember that you opened the header to modify a request.

__13. Close any open web browser.

Part 6 - Review

Message integrity is used to implement the non-repudiation feature. That is a legal term which means, a party sending a message can not deny that it had sent that message. In addition, it can not claim that a third party had altered the message.

Enabling message integrity is the first step in setting up non-repudiation. The provider will validate the request and the consumer will validate the response. Any man in the middle attack will be immediately detected. However, to resolve any disputes between business at a later date, you need to also save the message, its signature and sender's certificate in an audit log. In case of a dispute, the message should be validated again.

In this lab, we enabled message integrity. We also attempted a man in the middle attack which was successfully thwarted.

We also learned a lot of policy set and policy binding. They allow us to change the client server communication without changing any code.