

Integrate a secured Web service into a J2EE project

Skill Level: Advanced

[Frank Franco](#)
Software Engineer

[Qun Zhou](#)
Software engineer

06 Jul 2005

This tutorial demonstrates how you can integrate a secured Web service into an IBM(R) WebSphere(R) Studio Application Developer J2EE project with a step-by-step Enterprise JavaBeans (EJB) project sample application.

Section 1. Before you start

About this tutorial

This tutorial demonstrates how you can integrate a secured Web service into an IBM(R) WebSphere(R) Studio Application Developer J2EE project with a step-by-step Enterprise JavaBeans (EJB) project sample application.

This tutorial assumes that you understand the basic ideas behind a Web service, are familiar with using WebSphere Studio Application Developer (Application Developer), and have the need to integrate a secured Web service into your J2EE application.

In this tutorial, you will learn:

- How to deploy a secured Web service to an EJB project.

- How to configure Web service transportation level security: Basic Authentication.
- How to add more EJBs to access the secured Web service.
- How to set-up time-out parameters for the Web service.

All the steps are tested on Application Developer V5.1.2.

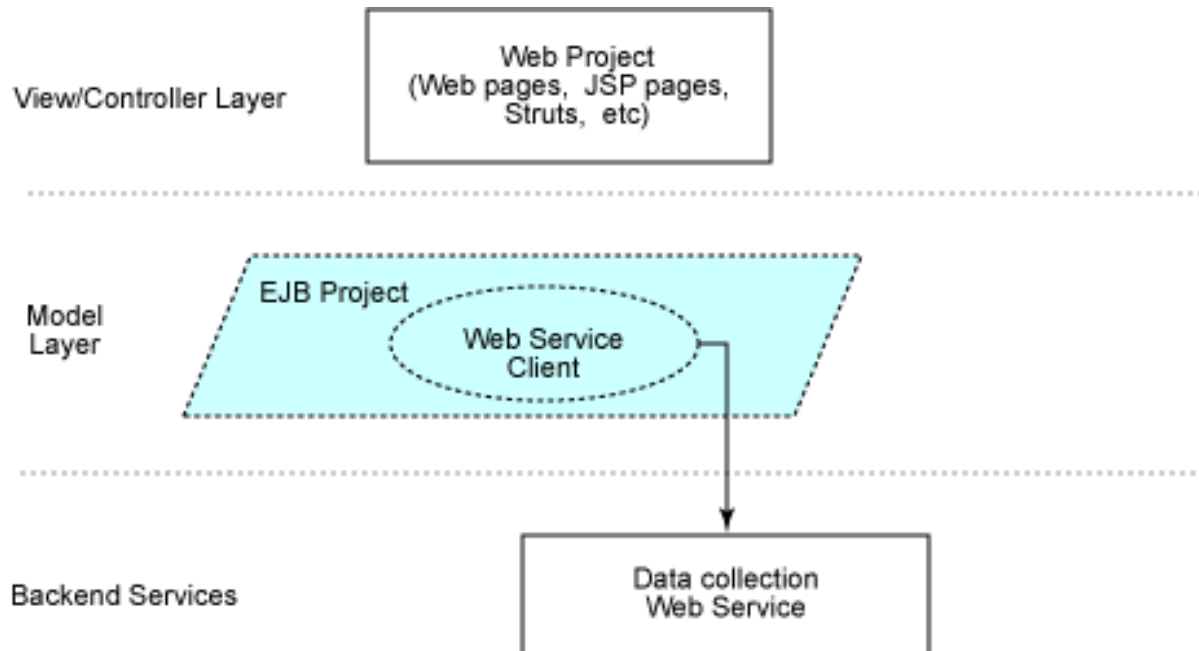
Section 2. Background information

Example project introduction

Our example application is called Seeker. Seeker is responsible for discovering vulnerabilities on a network and ultimately being able to take action on infected devices. Although Seeker already has integrated many corporate sources to accomplish its goal, a new one which is implemented as a Web service is now available. This tutorial details how to integrate this secured Web service into the existing application.

The Seeker application uses a MVC (Model View, Controller) architecture. Because of this, we want to integrate the Web service into the Model Layer opposed to the View Layer. For our example, the SeekerWeb project contains the View and Controller Layers, and the SeekerSearchEJB project contains the Model Layer, or the business layer which contains the back-end processing. The figure below illustrates this:

Figure 1. MVC Design



What we explain in detail is how to integrate the Web service client into the EJB project.

For secure Web services, Application Developer helps remind us that:

"webservicesclient.xml, ibm-webservicesclient-bnd.xmi and ibm-webservicesclient-ext.xmi are used only when running Web service clients in a container-managed environment, and are not used when running Web service clients in a non-managed environment. Therefore secured Web services can only be accessed from Web service clients running in a container-managed environment, otherwise the required security information is unavailable to the client."

Deliverables from the service provider

In our example, the Web service developers provided the following:

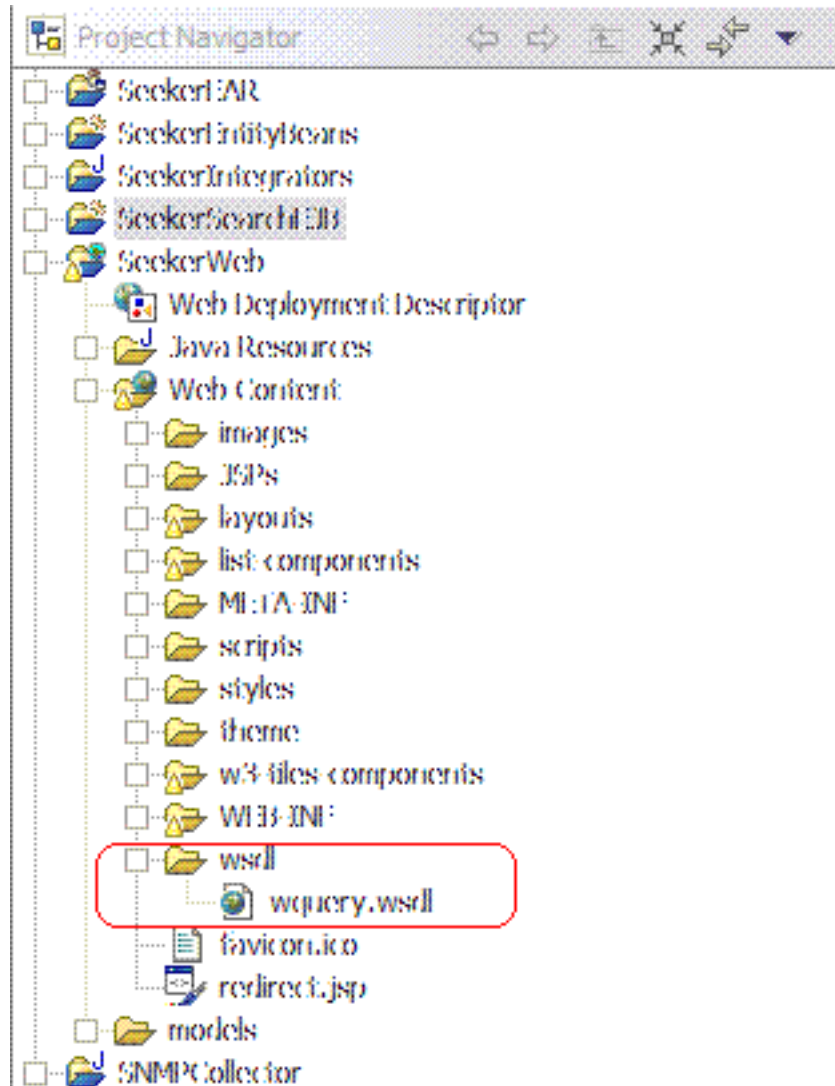
1. One Web Services Description Language (WSDL) file: wquery.wsdl
2. Three helper classes which are used to parse the returned xml content from the Web service

Section 3. Integration steps

Deploy a secured Web service into an EJB project

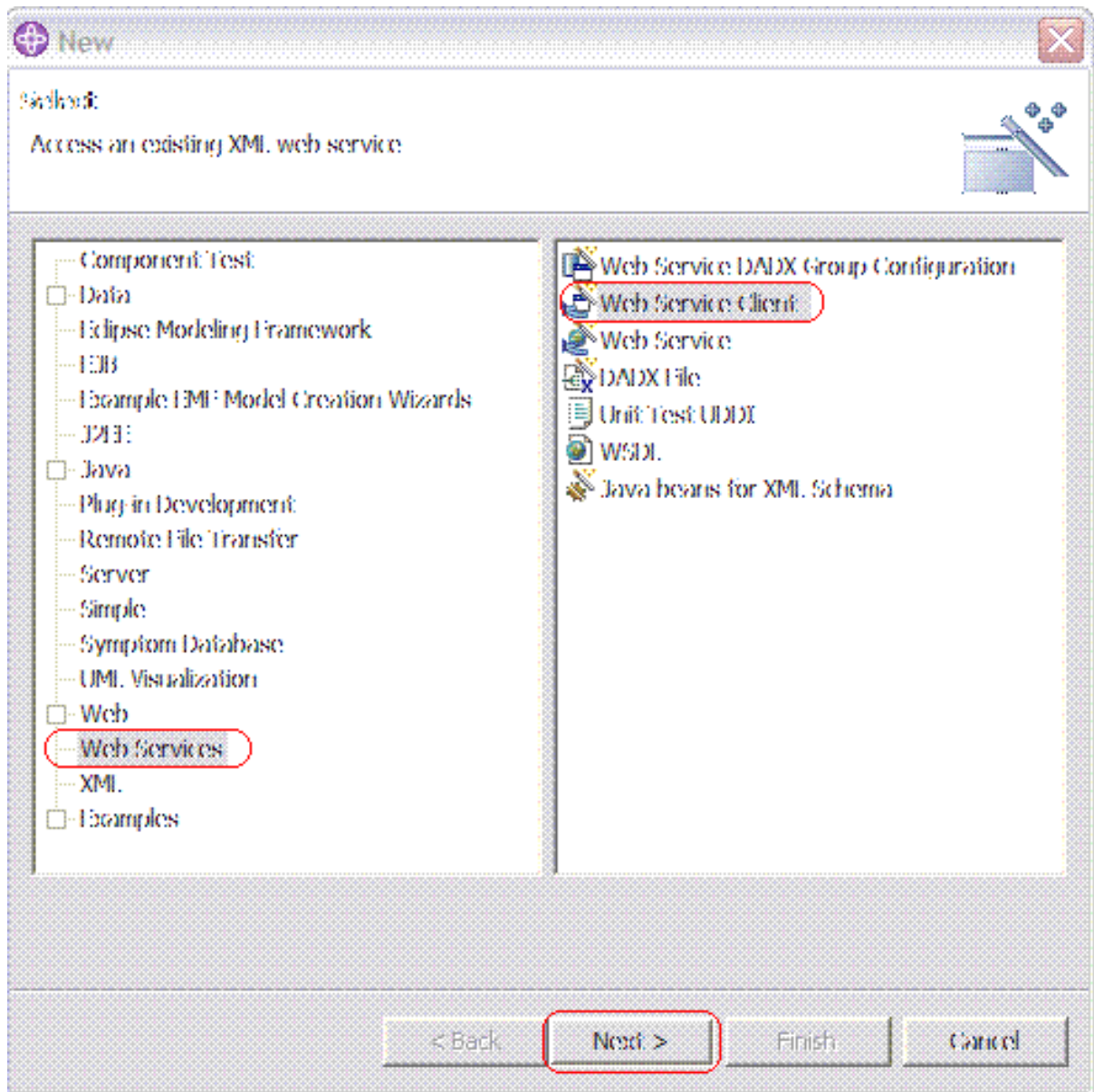
Step 1: In the Web project *SeekerWeb*, create a WSDL folder under *Web Content* and import the WSDL file (*wquery.wsdl*) into the newly created folder. You can put the WSDL file into any project inside this application. It can be removed after the WSDL file is successfully deployed into the desired destination.

Figure 2. Import the WSDL file into the Web project



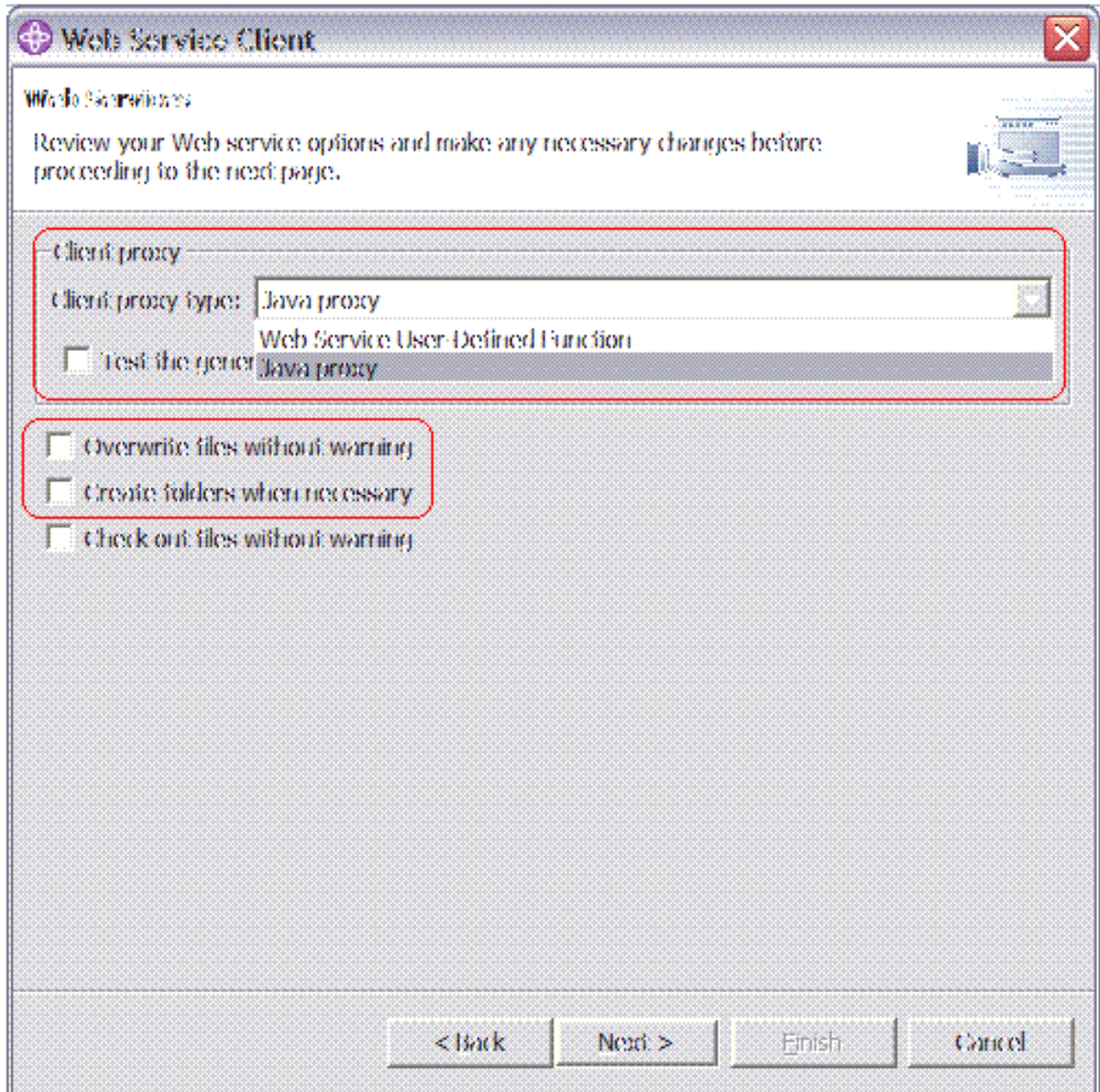
Step 2: Right click the EJB project, in our case it's *SeekerSearchEJB*. Choose **New > other**, select **Web Service** on the left side, and select **Web Service Client** on the right side. Then click **Next**.

Figure 3. Create a new Web service client



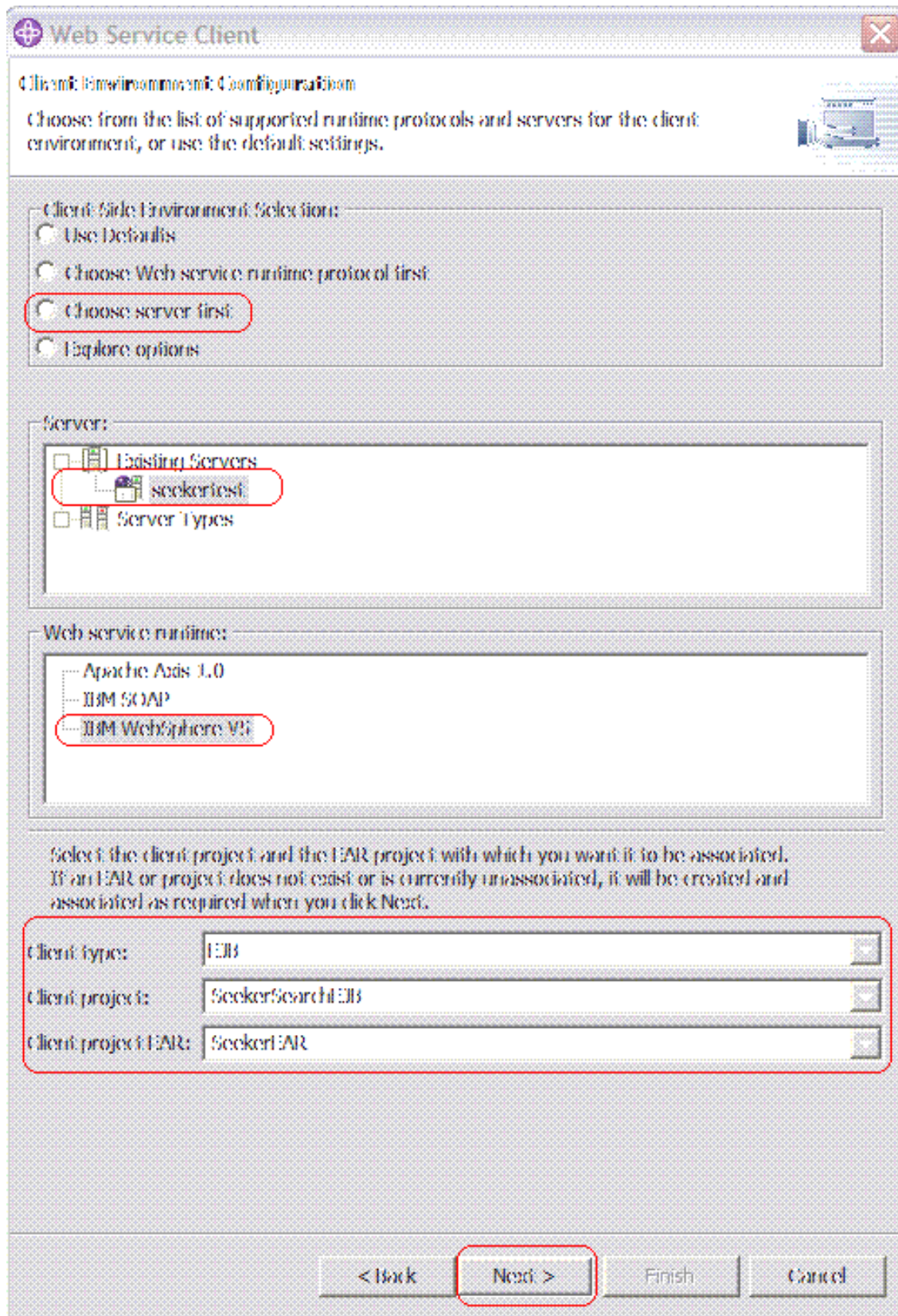
Step 3: There are two options for the Client proxy. For our purposes, choose **Java proxy**. Check **Overwrite files without warning** and **Create folders when necessary** if they are not checked. Then click **Next**.

Figure 4. Set the Web service client proxy type



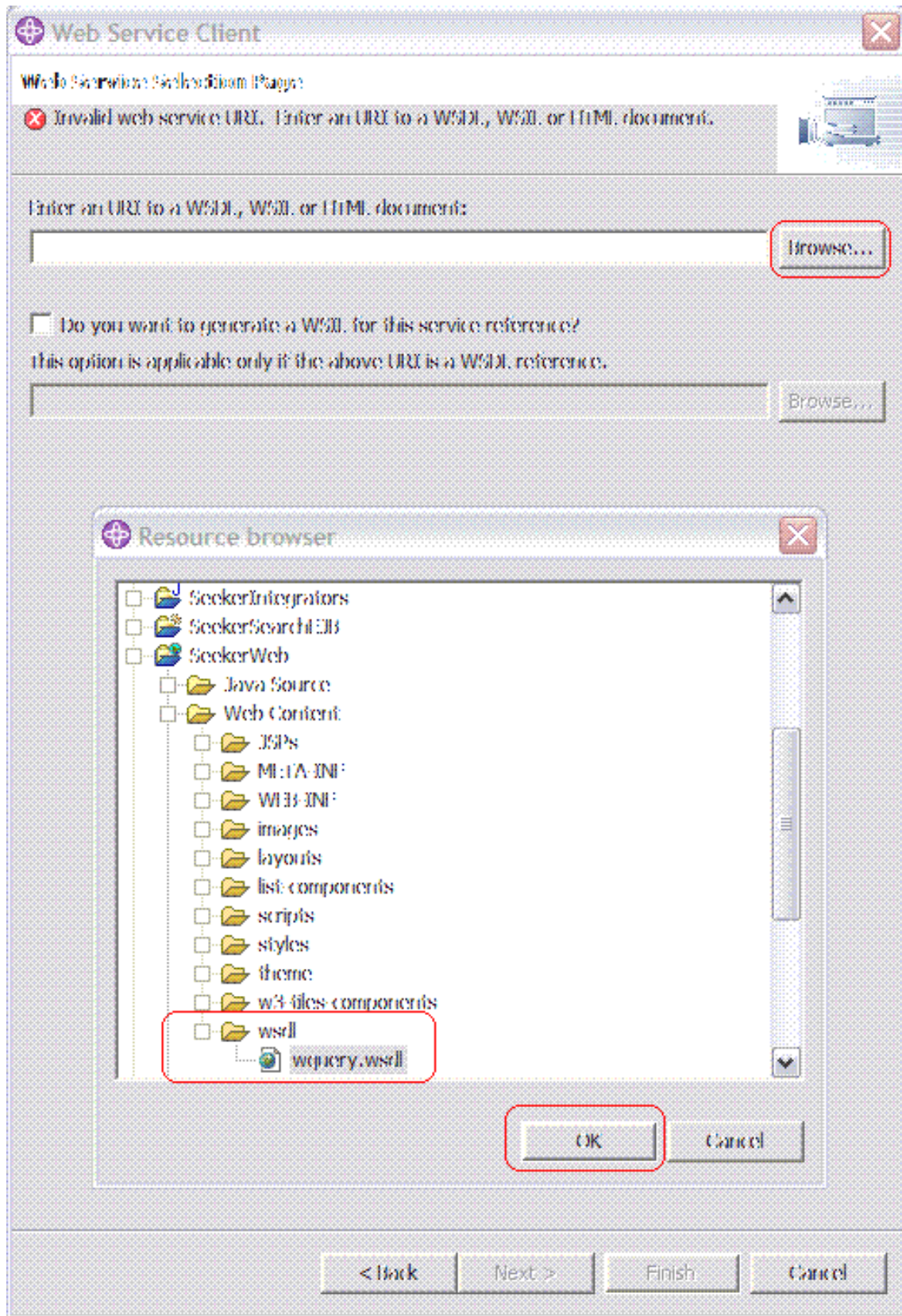
Step 4: On this panel, click **Choose server first** for Client-Side Environment Selections and make sure that the Server and Web service run-time are correct. Accept the default value for Client type, Client project, and Client project EAR. Then click **Next**.

Figure 5. Set the client environment



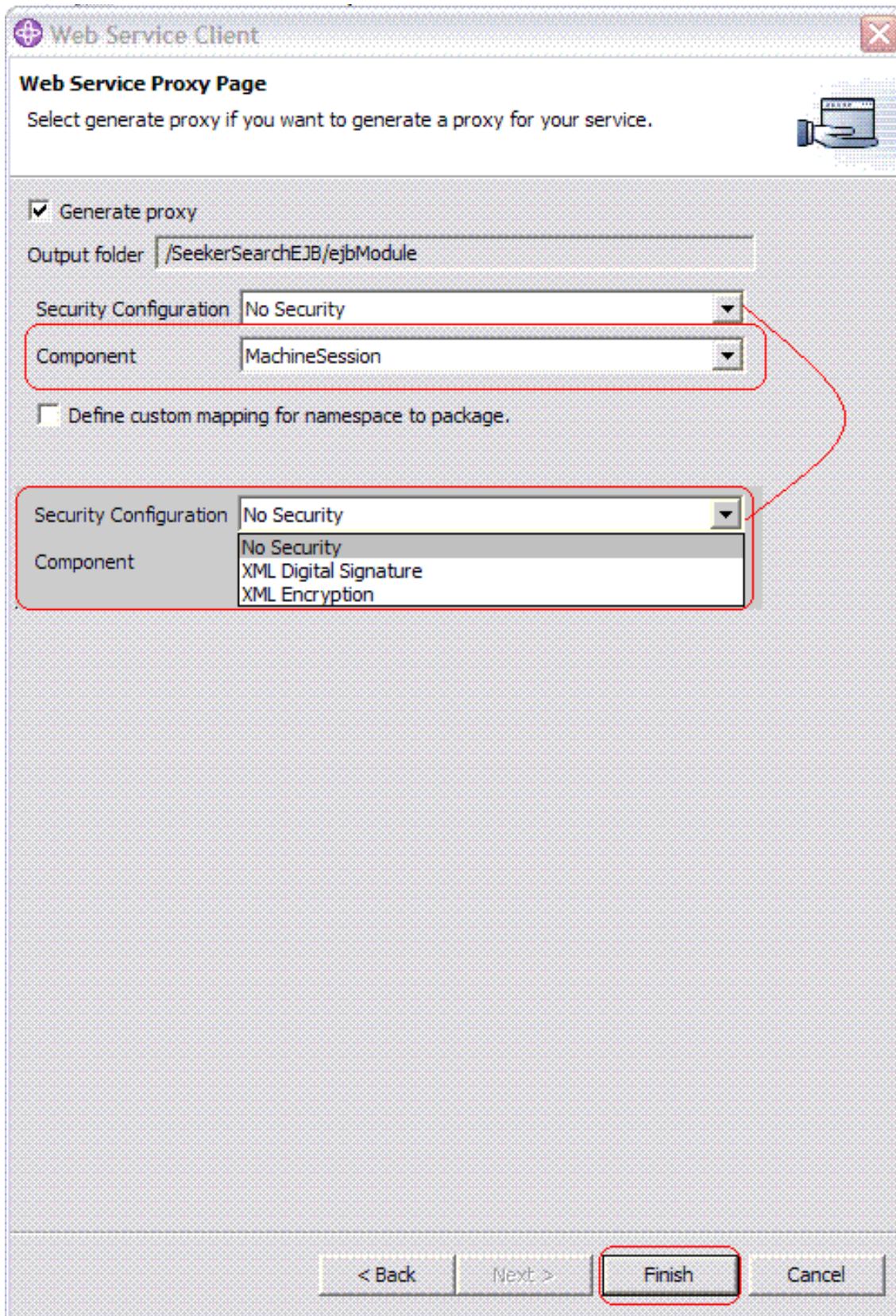
Step 5: Click **Browse** to select the deployed WSDL file. Then click **Next**.

Figure 6. Select the WSDL file



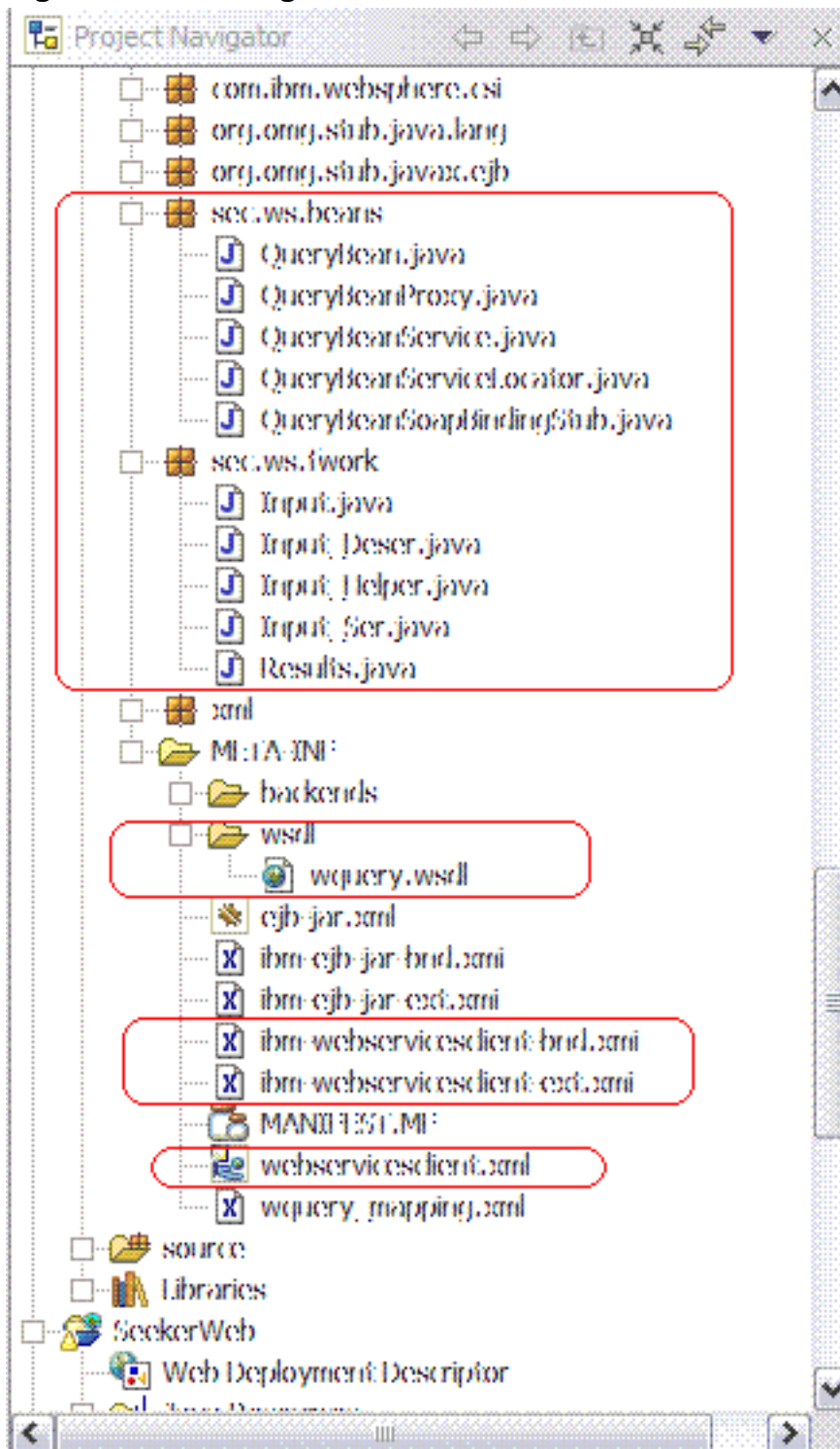
Step 6: This specific Web service uses transportation-level security (Basic Authentication) so take the default *No Security* choice for Security Configuration which is setting message-level security. Also, choose the EJB that makes the Web service call, which, in our case, is *MachineSessionBean* . Here you only can pick one bean. If you have several beans which need access to the secured Web service, refer to the section [Add additional session beans to access the secured Web service](#).

Figure 7. Select the session bean



Step 7: Then click **Finish**. This generates the client-side stub classes. You can check the SeekerSearchEJB and ejbModule folders for the new proxy classes.

Figure 8. View the generated stub classes



Security settings

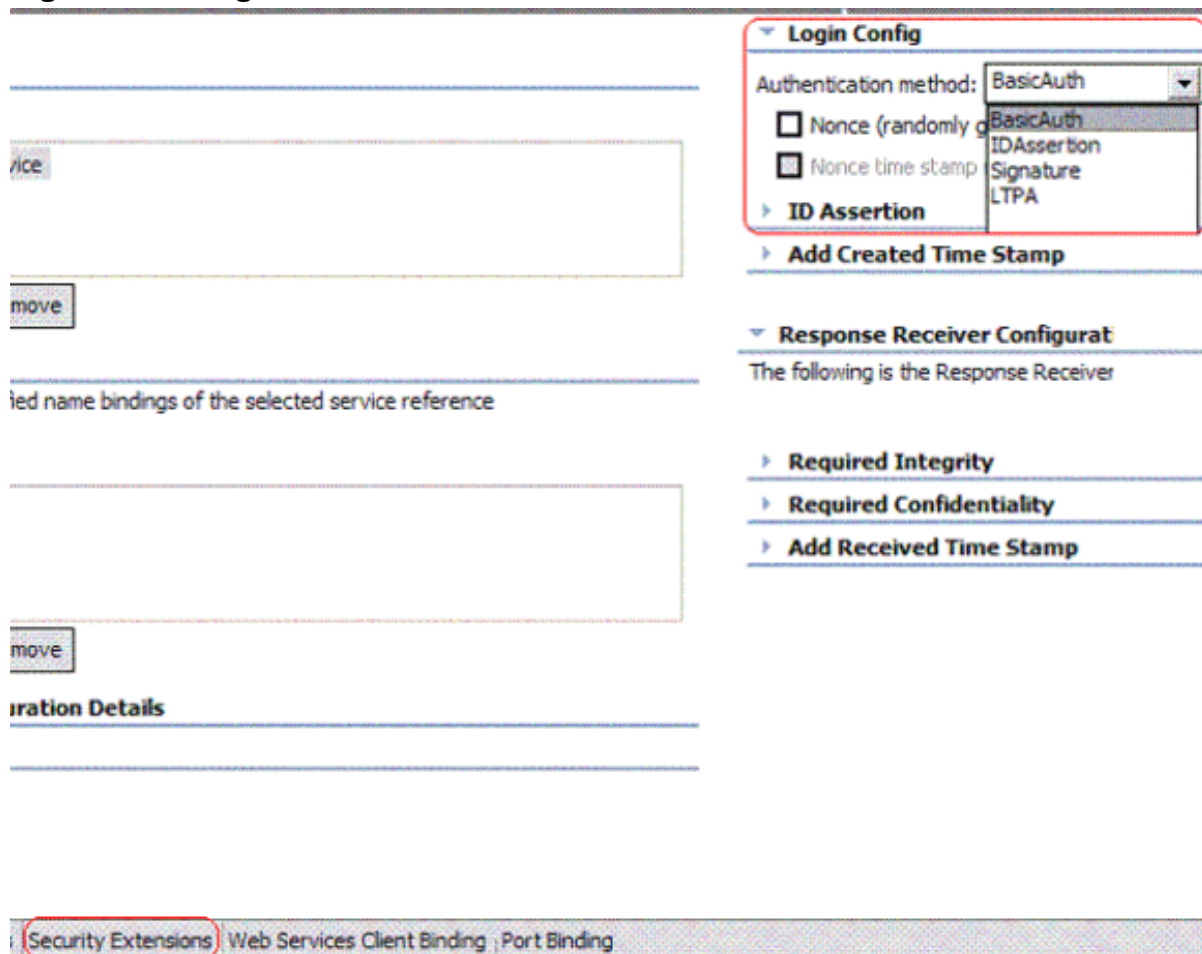
You need to consult the Web services provider to find out how to set the security. Different implementation may result in different processes. In our case, this Web service uses a user authentication method. The container (Web or EJB) activates the specified authentication mechanism when the client tries to access the protected resource.

Step 1: Open the file `webservicesclient.xml`, which is under the directory `SeekerSearchEJB > ejbModule > META-INF`.

Step 2: In the *Service Reference* panel, select the EJB you specified in the last section in Step 6. In our case, this is *MachineSessionBean*.

Step 3: Go to the *Security Extensions* tab. Expand the *Login Config* section. Set the authentication method to **BasicAuth**.

Figure 9. Configure the authentication method



Step 4: Now, go to the *Port Binding* tab. Expand the *Security Request Sender Binding Configuration* section. Expand the *Login Binding* section. Click **Enable** to open a login binding dialog box.

Figure 10. Enable the authentication setting



Step 5: In this dialog box, select the correct authentication method as **Basic Auth** and the callback handler field value. Here we choose `com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler`, since we don't want a dialog pop-up during the Web service call. Also, enter a valid user ID and password for your Web service. Finally, click **OK** to close the dialog box.

Figure 11. Set the user ID and password

Login binding dialog

Authentication method: Basic Auth

Token value type:

URI:

Local name:

Callback handler: com.ibm.wsspi.wsssecurity.auth.callback.NonPrompt

Basic authentication:

User ID: qzhou@us.ibm.com

Password: *****

Property:

Name	Value

Add Remove

OK Cancel

Step 6. Save the webservicescient.xml file. Leave the file open if you are interested in the next section on adding additional session beans.

At this point, you successfully deployed a secured Web service into your EJB project. You can call the Web service from your session bean, which in our case is *MachineSession*.

Add additional session beans to access the secured Web service

In a real development environment, having only one session bean that can access the Web service might not be sufficient. Until now, in our example application, we only have one session bean *MachineSession*, which can access the corporate data from the secured Web service. What if another session bean, named *DeviceSession*, also needs to access the same data source? In this section, we guide you through adding additional session beans to access the secured Web service. For your convenience, you can copy the configuration of the session bean you already set from the above two sections.

Step 1. Go back to the *Service References* panel, click **Add** in the *Component scoped references* section, which adds an editable entry called *Port component reference* into the list.

Figure 12. Add a session bean which can access a secured Web service

Service References

Editor for service references

Component-scoped references

The following are component-scoped references in this Web service client deployment descriptor

```

PersonSession
IncidentSession
MachineSession
Port component reference
<unqualified>
  
```

Add

Remove

Service references

The following are Web services referenced by this module:

```

service/Service reference
  
```

Add

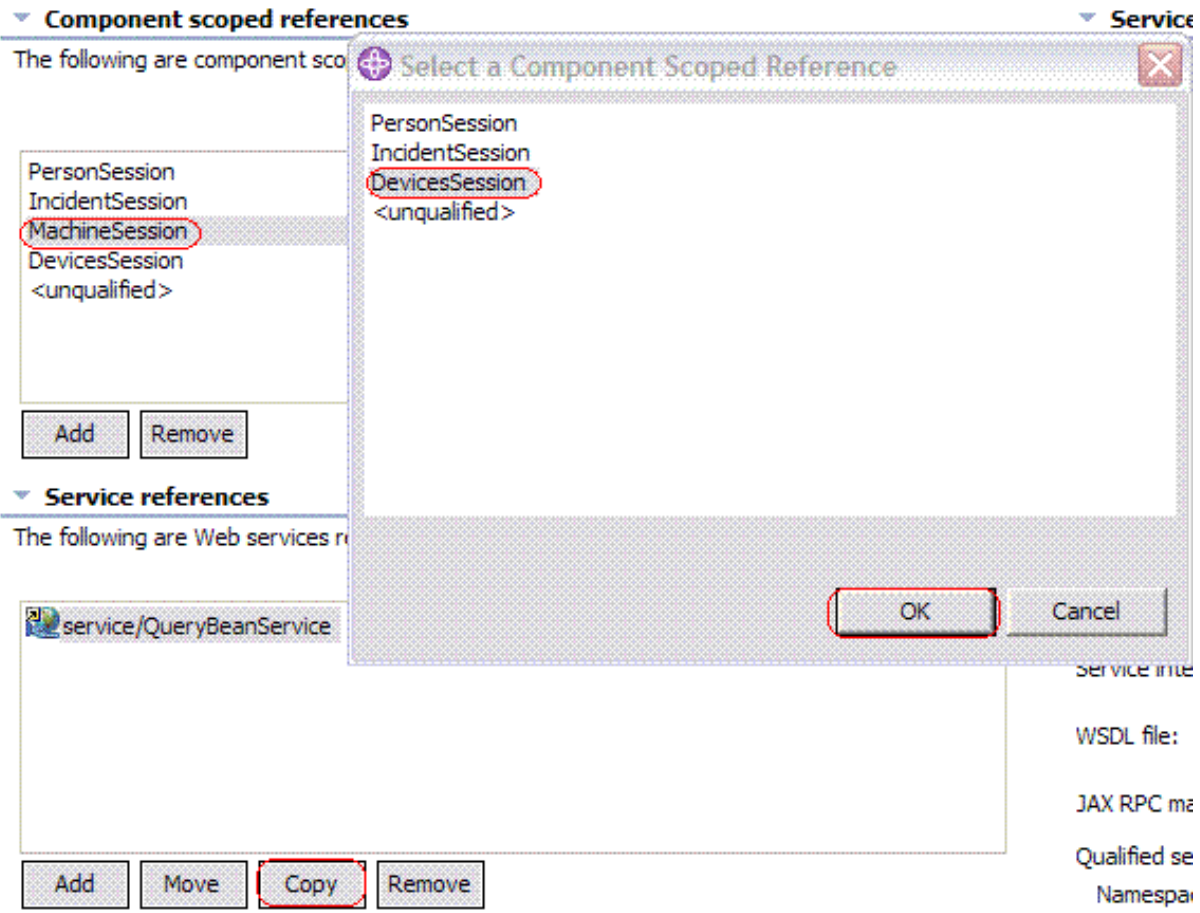
Move

Copy

Remove

Step 2. Change the *Port Component reference* to the session bean name, for example *DevicesSession*. Then, remove the default service references *service/Service reference*. From the *Component scoped references* list, select a session bean which has been successfully configured using all of the steps in the above sections. In our case, we choose *MachineSession* and click **Copy** under the *Service references* section.

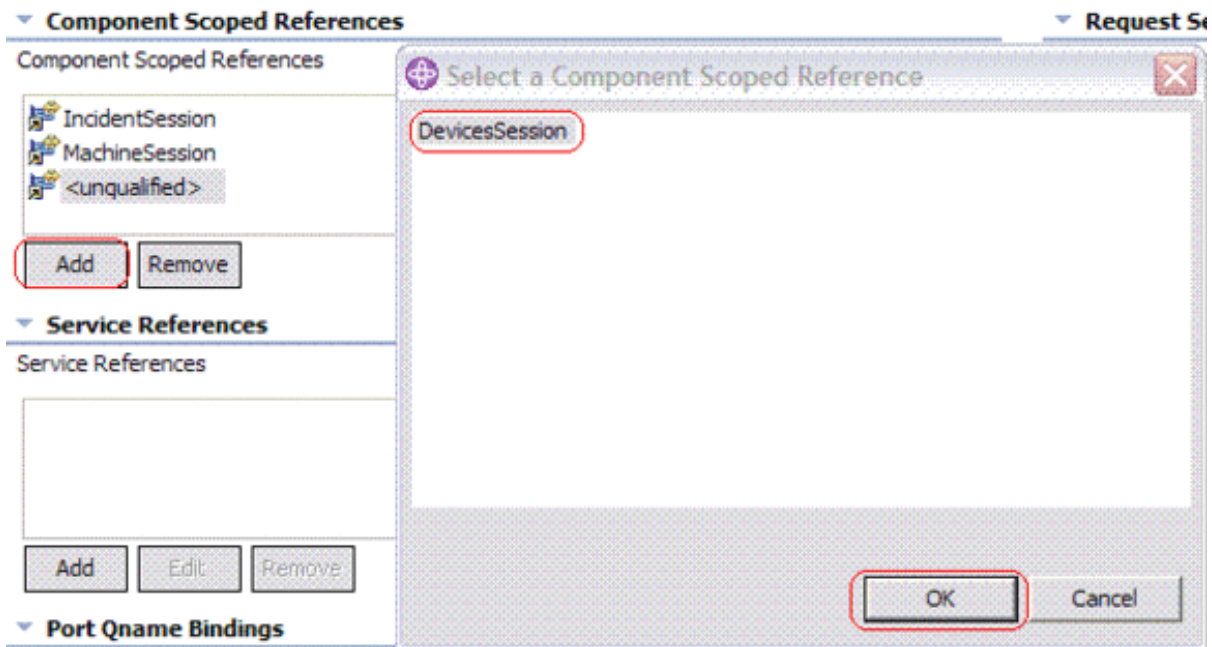
Figure 13. Select the newly added session bean



This brings up a dialog box which lists all of the session beans in the *Component scoped references* except for the source session bean, from where you can pick the target session bean -- in our case, *DevicesSession*. Then click **OK**, so that all of the information from *MachineSessionBean* is copied to *DevicesSessionBean*.

Step 3. Go to the *Security Extensions* tab and click **Add** in the *Component Scoped References* section, which brings up a dialog with the session bean without security configuration.

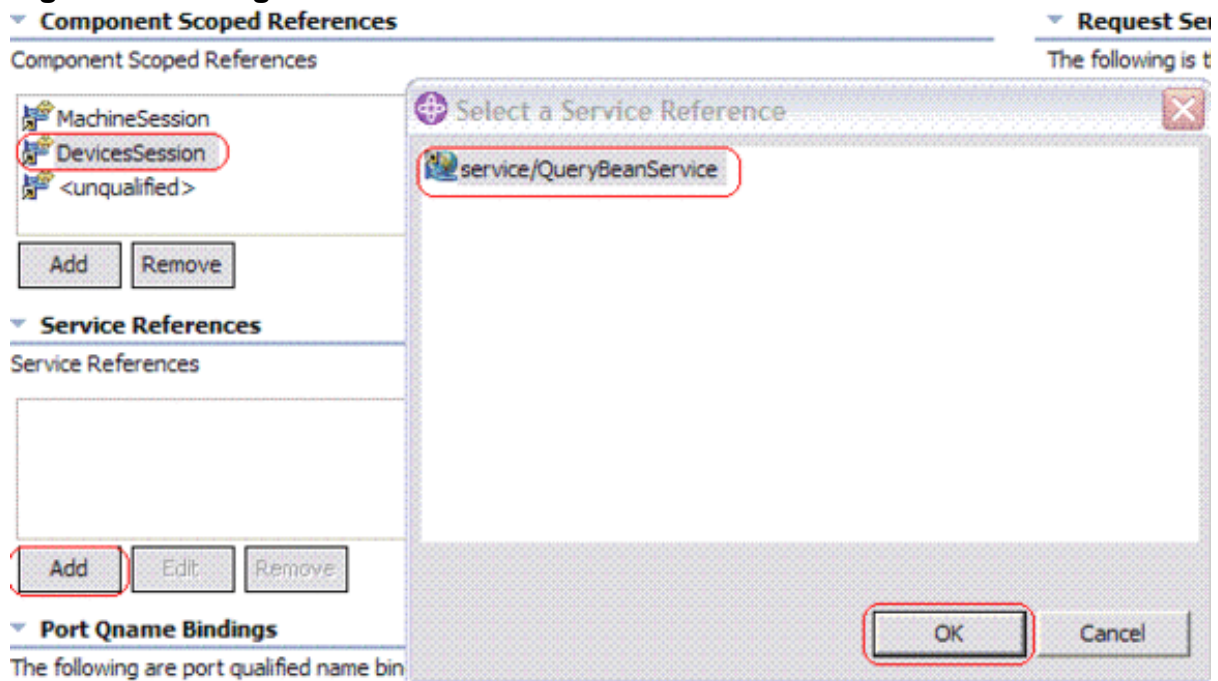
Figure 14. Configure the security extension



Select the newly added session bean and click **OK** to add it to the *Component Scoped References* list.

Step 4. Select the newly added bean *DevicesSession* and click **Add** in the *Service References* section, which brings up a dialog that lists the available service reference.

Figure 15. Configure the service reference



Step 5. Repeat Step 3 from the [Security settings](#) section to set the *Authentication method* to **Basic Auth** in the *Login Config* section.

Step 6. Go to the *Web Services Client Binding* tab and repeat steps 3 and 4 in this section.

Step 7. Go to the *Port Binding* tab and repeat steps 4 and 5 from the [Security settings](#) section.

Step 8. You can repeat steps 1 through 7 from this section to add as many session beans as you would like.

Step 9. Save and close the `webservicescient.xml` file.

Example code

At this point, you've successfully deployed a secured Web service into your EJB project, configured the security, and added as many session beans as required to access the secured Web service. To actually access the Web service from the code in your session bean, you can instantiate a proxy and then call the proxy's method(s) to get the result. For example:

```
...
//Get a WebService proxy
QueryProxy qproxy = new QueryProxy();
//Set up the input parameter
Input input = new Input();
input.setInput(.);
//Make a call and get the result
String result = qproxy.callmethod(input);
//process the result
...
```

Configure a time-out for a Web service

In an ideal world, you don't need to worry about a time-out. But in reality, the server which holds the Web service might not be available all the time, or sometimes the server itself has performance issues. For these reasons, it is very important to detect time-outs. If it happens, you might want to have a back-up path to get your job done. This might involve trying other servers, other data sources, and so on.

One very simple solution is to set a socket option. The time-out value can be set in a properties file. To use this solution, the client needs to run in a managed environment (like inside of an application server). Once you have the stub instance, you can call the `stubInstance.setTimeout()` method and pass-in an integer. The following example shows you how to set this option.

```
...
//Get timeout value from property file
String timeoutStr = rb.getString( "timeout" );
int timeout = Integer.parseInt(timeoutStr);
((QueryBeanSoapBindingStub)QueryBean).setTimeout(timeout);
...
```

Section 4. Summary

In this tutorial, you learned how to deploy a Web service in a multi-tiered application environment. You also learned how to configure the security and add references from more than one EJB to the Web service. The tutorial also discussed how to use the Web service's time-out parameter in case the Web service is not responding quickly.

Resources

Learn

- Get more information about a secured Web services run-time environment at the [WebSphere combined information center](#)
- Learn more about addressing security within a Web services environment in the whitepaper "[Security in a Web Services World: A Proposed Architecture and Roadmap](#)" (developerWorks, April 2002).
- Learn how to create a portable end-to-end Web service in multiple J2EE environments with the IBM WebSphere Studio Application Developer 5.1.1 Web Service wizard in the tutorial "[Create a portable Web service in multiple J2EE environments](#)" (developerWorks, September 2003).
- Find security-related [specifications and standards](#) on developerWorks.

Get products and technologies

- Get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®. You can [download evaluation versions](#) of the products at no charge, or select the Linux® or Windows® version of developerWorks' [Software Evaluation Kit](#).

Discuss

- [Participate in the discussion forum for this content.](#)

About the authors

Frank Franco

Frank Franco is a Software Engineer at the IBM Research Division. He has 10 years of experience in Java development and has architected many applications ranging from network remediation solutions on the Web, to end-to-end monitoring of e-commerce services.

Qun Zhou

Qun Zhou is a Software Engineer at IBM Research Division, where she is working on distributed virus detection and vulnerability remediation system. She once contributed to projects such as IBM Rational Application Developer V6.0 (WebSphere Studio Application Developer V5.12), IBM WebSphere Portal Server versions 5.0 and 5.1, Lotus

Workplace versions 2.0 and 2.5, and IBM CNN media content management. For technical questions or comments about the content of this tutorial, contact the author, Qun Zhou, at qzhou@us.ibm.com.