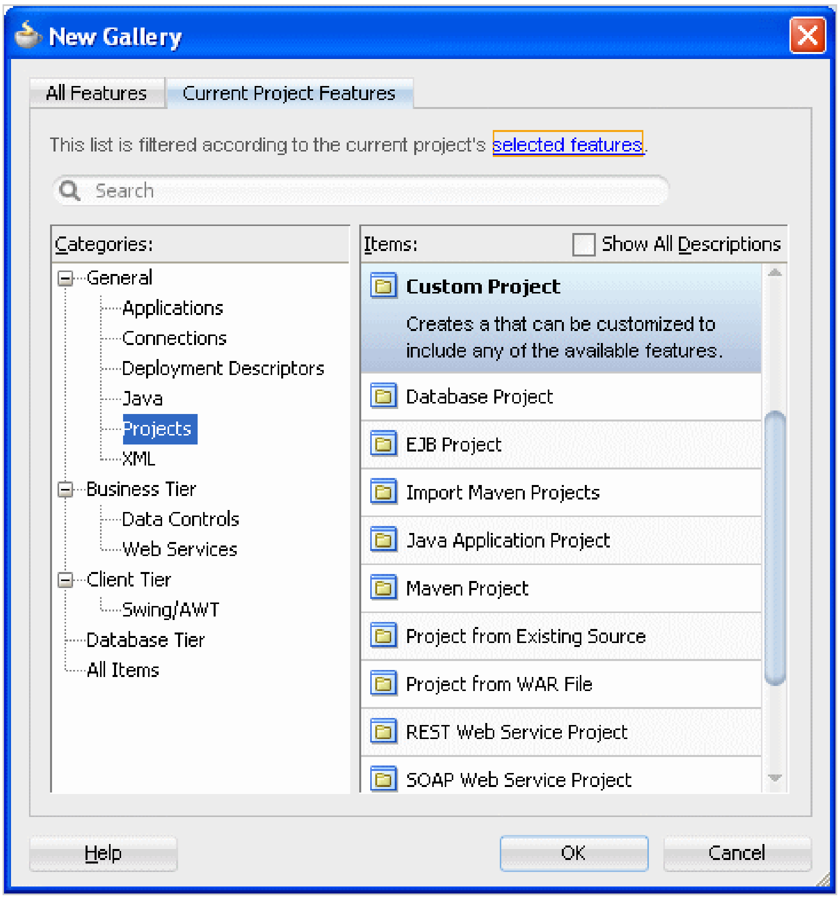
Part 2: Building a Declaratively-Driven POJO Web Service

Any Java method can be published as a Web Service. JDeveloper provides wizards that take a Java class and its methods and creates a Web Service from that class.

In this section you create a new project and Java class, just as you did earlier. In this scenario, rather than manually adding annotations to create the web service, you use a wizard. The wizard creates all the necessary annotations to enable the class as a web service. Once the wizard steps are complete, you test the web service using the HTTP Analyzer with the integrated server, just as you did before.

Step 1: Create a Plain Old Java Object (POJO)

1. Create a new empty project. Right-click the Annotation project node and select **New**. In the New Gallery, click the **All Features** tab and then select **General > Projects** in the Categories list and **Custom Project** in the Items list. Click **OK**.

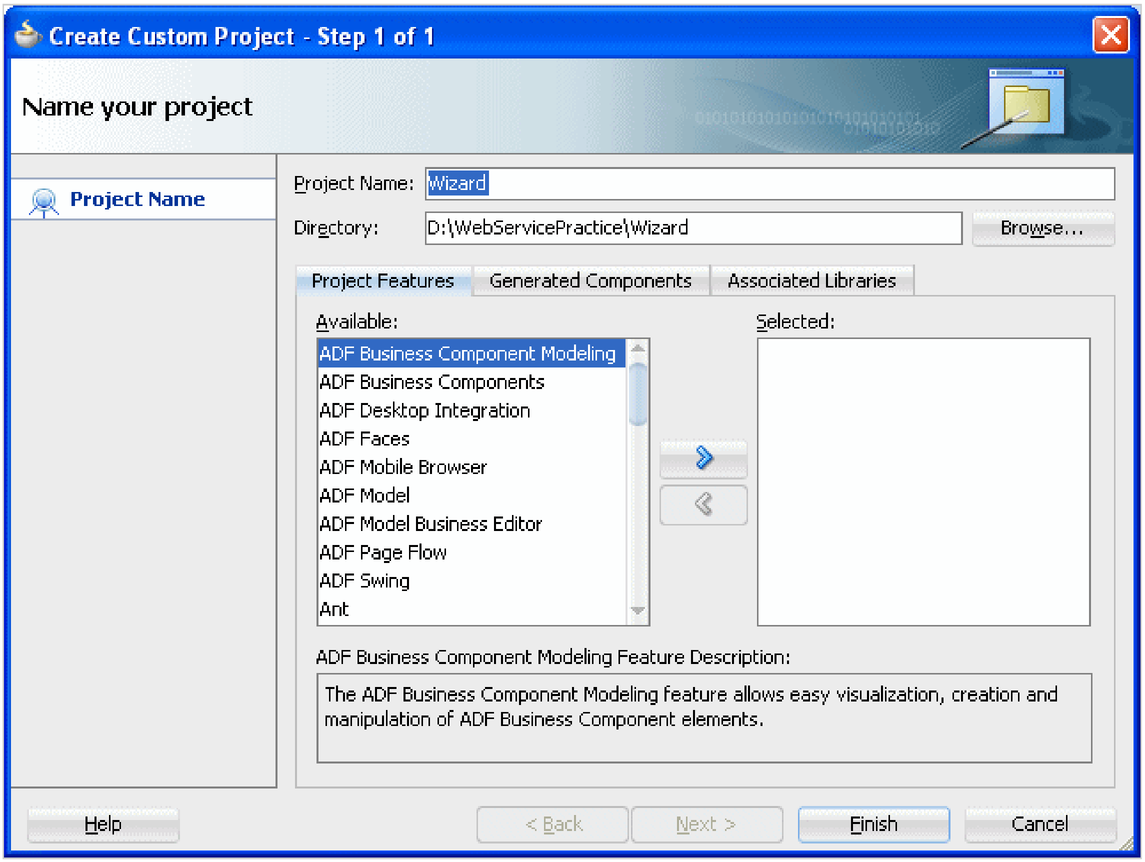


This list is filtered according to the current project's selected features.

Categories:
- General
  - Applications
  - Connections
  - Deployment Descriptors
  - Java
  - Projects
  - XML
- Business Tier
  - Data Controls
  - Web Services
- Client Tier
  - Swing/AWT
- Database Tier
- All Items

Items:
- **Custom Project**
  Creates a that can be customized to include any of the available features.
- Database Project
- EJB Project
- Import Maven Projects
- Java Application Project
- Maven Project
- Project from Existing Source
- Project from WAR File
- REST Web Service Project
- SOAP Web Service Project

2. In the **Create Project Dialog**, name the project **Wizard** and click **Finish**.



Project Name: Wizard

Directory: D:\WebServicePractice\Wizard

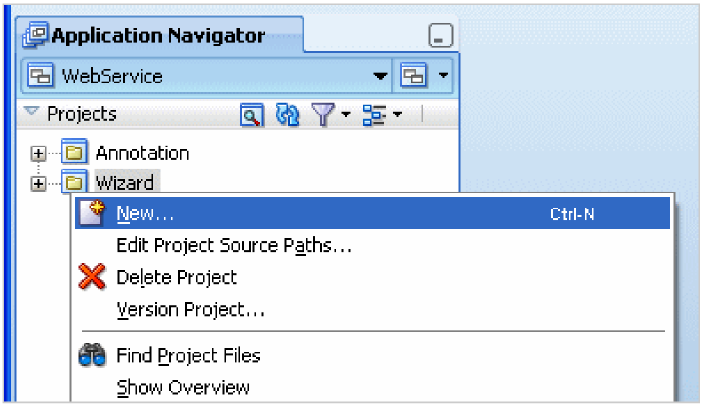
Available:
- ADF Business Component Modeling
- ADF Business Components
- ADF Desktop Integration
- ADF Faces
- ADF Mobile Browser
- ADF Model
- ADF Model Business Editor
- ADF Page Flow
- ADF Swing
- Ant

Selected:

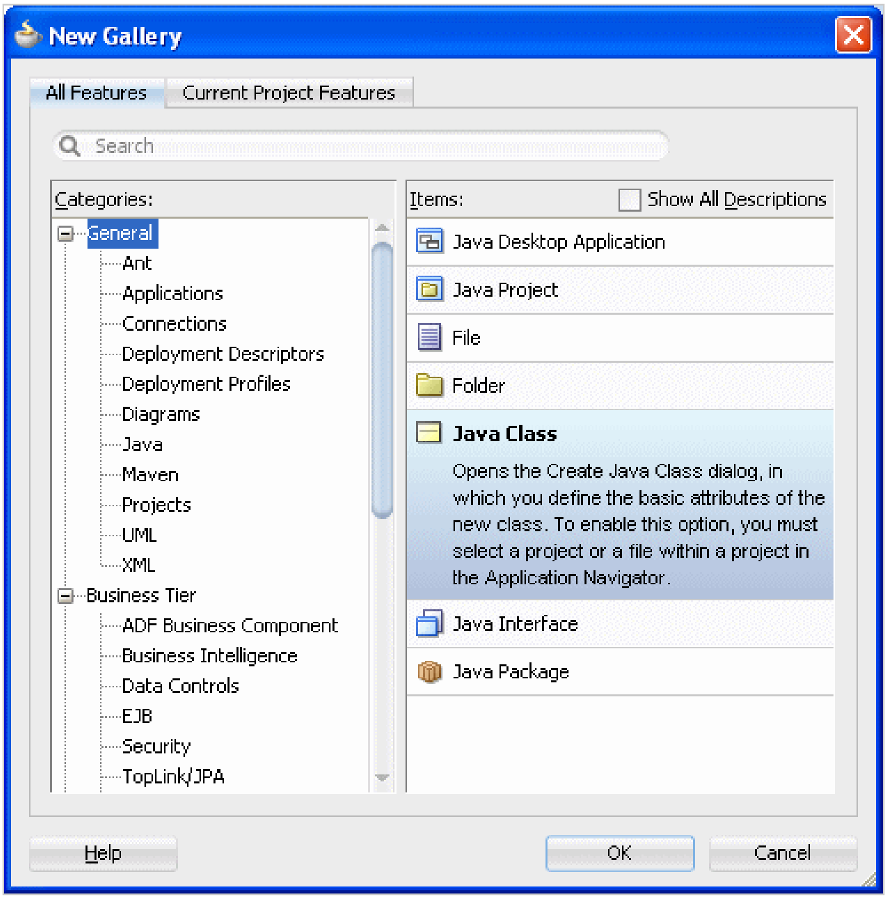
ADF Business Component Modeling Feature Description:

The ADF Business Component Modeling feature allows easy visualization, creation and manipulation of ADF Business Component elements.

3. In the Application Navigator, right-click the new **Wizard** project and select **New** from the context menu.



4. In the New Gallery select the **All Features** tab, then select **General** in the Categories list and **Java Class** from the Items list to create a new Java class.


New Gallery

All Features | Current Project Features

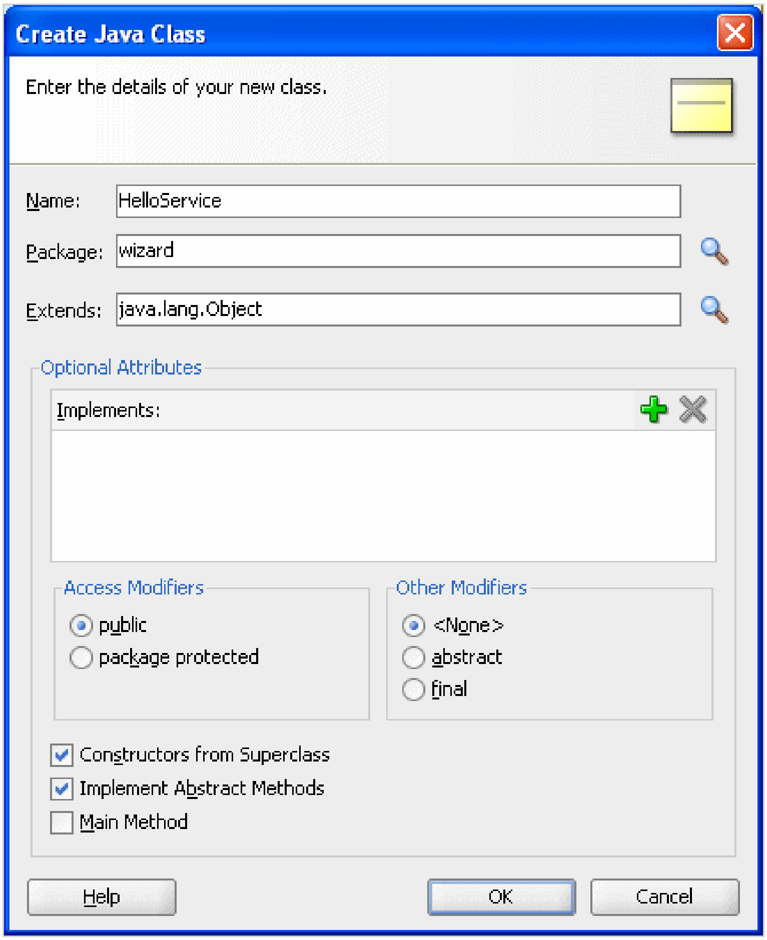
Search

Categories:
- General
  - Ant
  - Applications
  - Connections
  - Deployment Descriptors
  - Deployment Profiles
  - Diagrams
  - Java
  - Maven
  - Projects
  - UML
  - XML
- Business Tier
  - ADF Business Component
  - Business Intelligence
  - Data Controls
  - EJB
  - Security
  - TopLink/JPA

Items: ☐ Show All Descriptions
- Java Desktop Application
- Java Project
- File
- Folder
- **Java Class**
  Opens the Create Java Class dialog, in which you define the basic attributes of the new class. To enable this option, you must select a project or a file within a project in the Application Navigator.
- Java Interface
- Java Package

Help | OK | Cancel

5. Name the class **HelloService**. By default the package name should be the project name. Ensure that the package name is set to **wizard**. Leave the rest of the values at their defaults and click **OK** to invoke the Code Editor.


Create Java Class

Enter the details of your new class.

Name: HelloService
Package: wizard
Extends: java.lang.Object

Optional Attributes
Implements:

Access Modifiers
[✓ public]
package protected

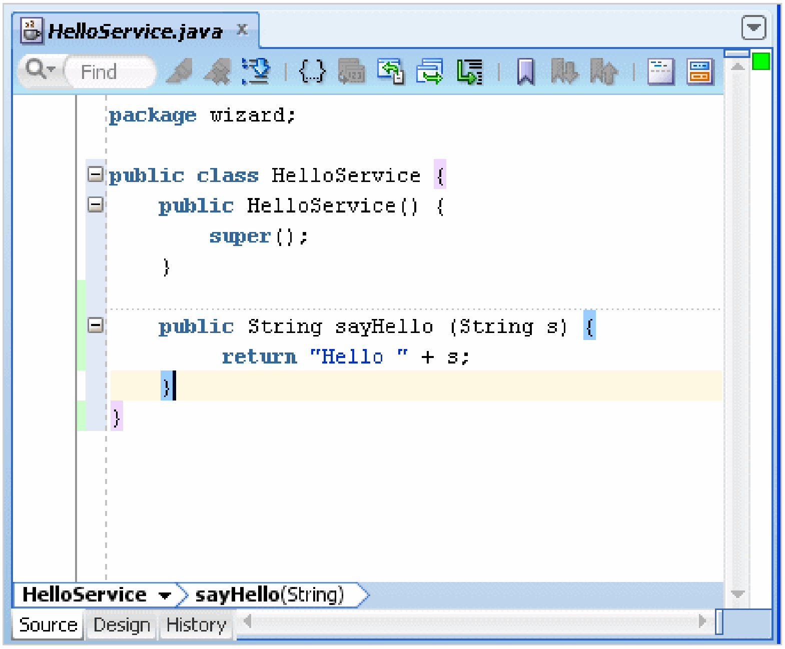
Other Modifiers
[✓ <None>]
abstract
final

☑ Constructors from Superclass
☑ Implement Abstract Methods
☐ Main Method

Help | OK | Cancel

6. Add the following method to the class:

```
    public String sayHello (String s) {
         return "Hello " + s;
    }
```



```
package wizard;

public class HelloService {
    public HelloService() {
        super();
    }

    public String sayHello (String s) {
         return "Hello " + s;
    }
}
```

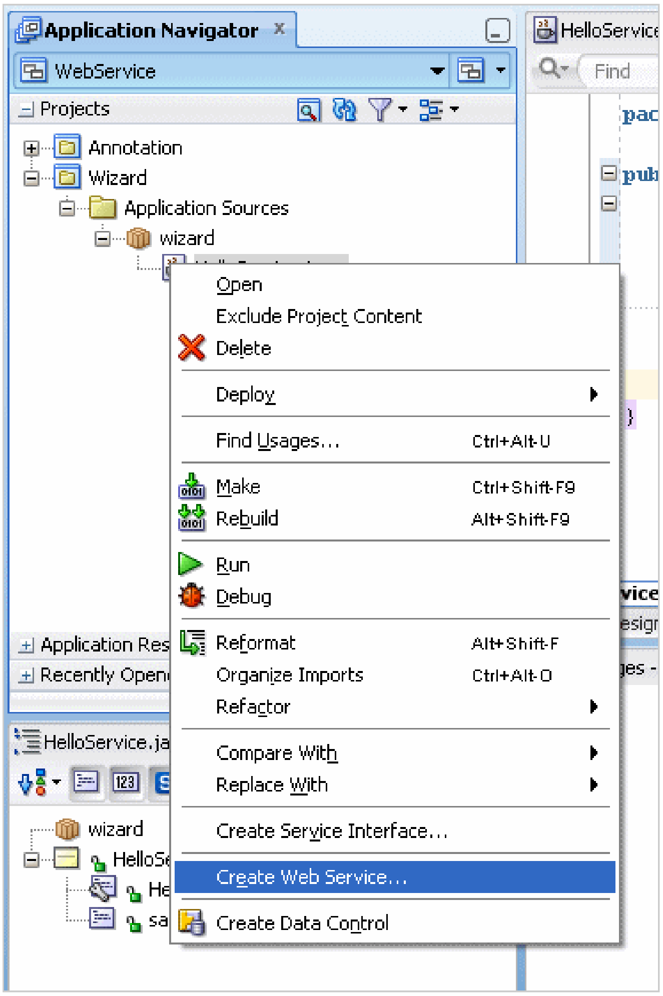
7. Click **Save All** to save your work.

   At this point you just have a class with some very simple business logic to return the word Hello followed by the argument value.

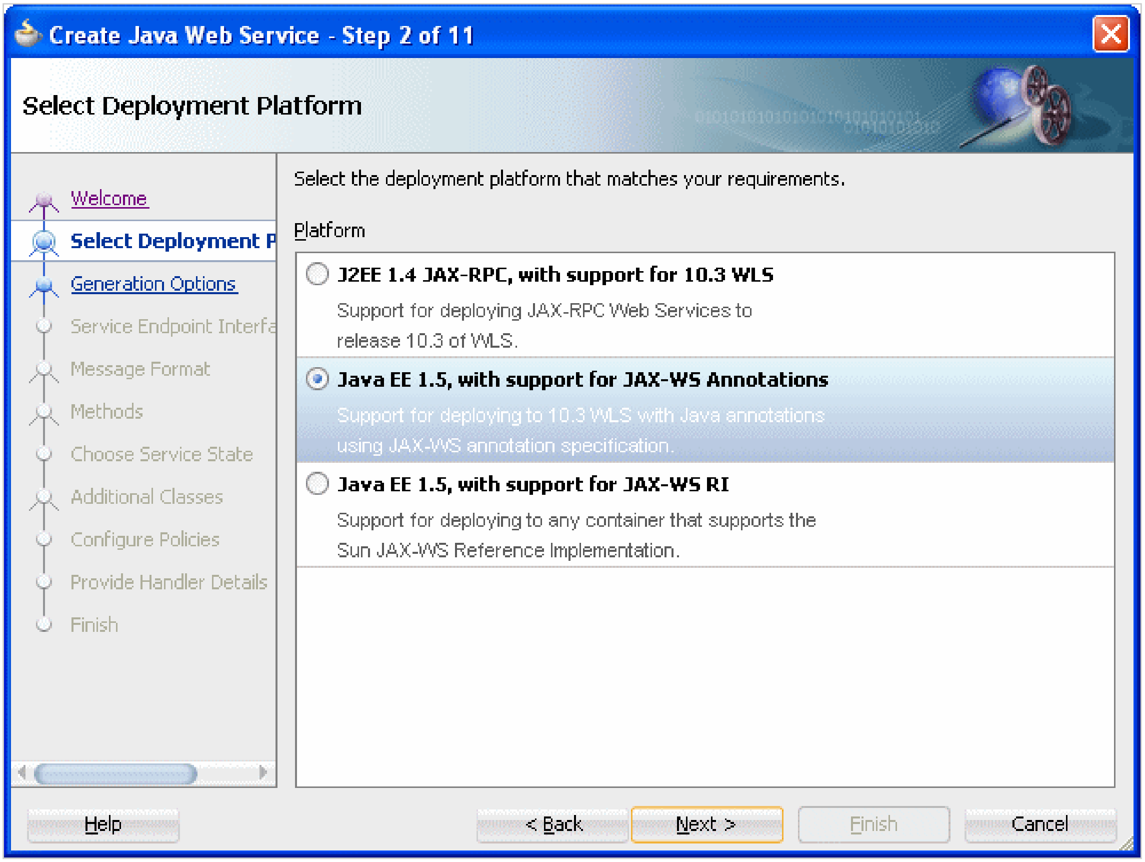
Step 2: Creating the Web Service Class

Now that you have a POJO (Plain Old Java Object), you can use JDeveloper to create the annotations necessary to publish any of the methods as web services.

1. In the Application Navigator, right-click the **HelloService.java** node and select **Create Web Service**. This starts the wizard to create the class as a web service.



2. In the Select Deployment Platform step of the Create Java Web Service wizard, ensure that the **Java EE 1.5, with support for JAX-WS Annotations** is selected as the deployment platform.

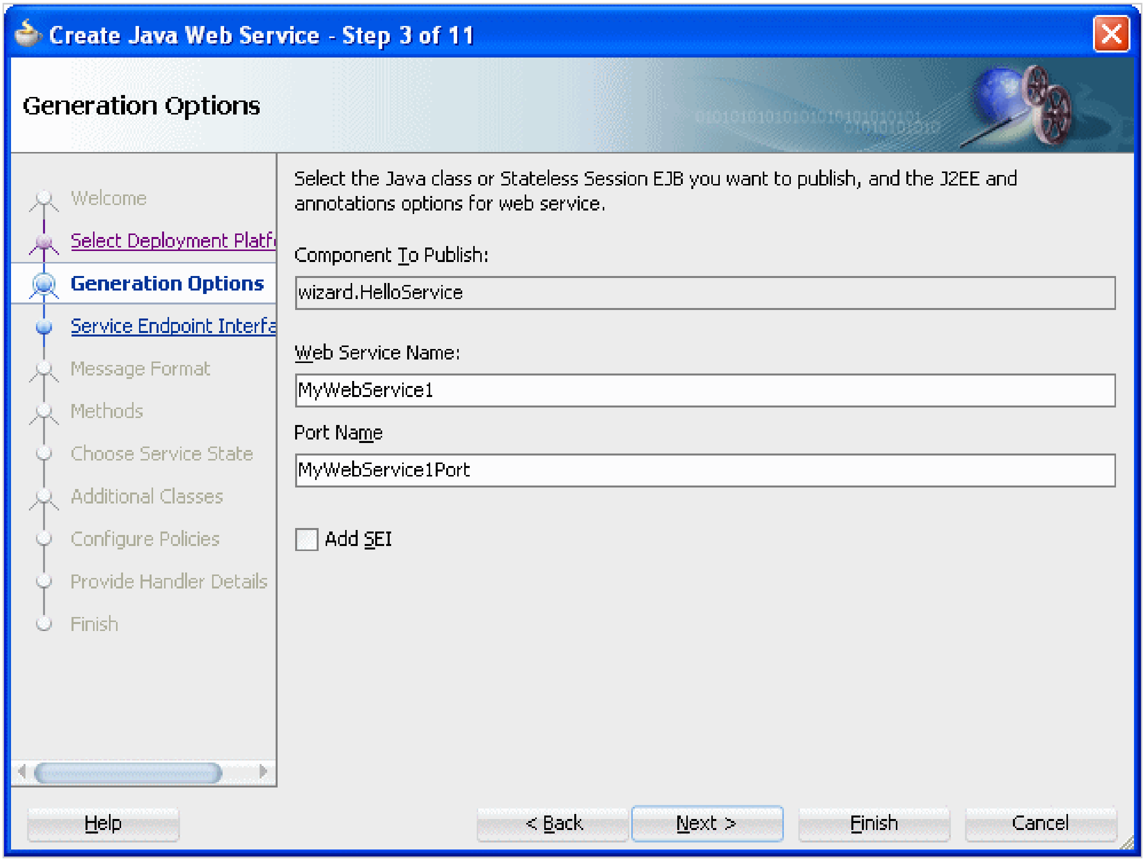


Select the deployment platform that matches your requirements.

Platform

- ( ) J2EE 1.4 JAX-RPC, with support for 10.3 WLS
  Support for deploying JAX-RPC Web Services to release 10.3 of WLS.
- (●) Java EE 1.5, with support for JAX-WS Annotations
  Support for deploying to 10.3 WLS with Java annotations using JAX-WS annotation specification.
- ( ) Java EE 1.5, with support for JAX-WS RI
  Support for deploying to any container that supports the Sun JAX-WS Reference Implementation.

3. Click **Next.**
4. In the Generation Options step of the wizard, type **MyWebService1** as the Web Service Name, and ensure that the Port Name is set to **MyWebService1Port**.



Select the Java class or Stateless Session EJB you want to publish, and the J2EE and annotations options for web service.

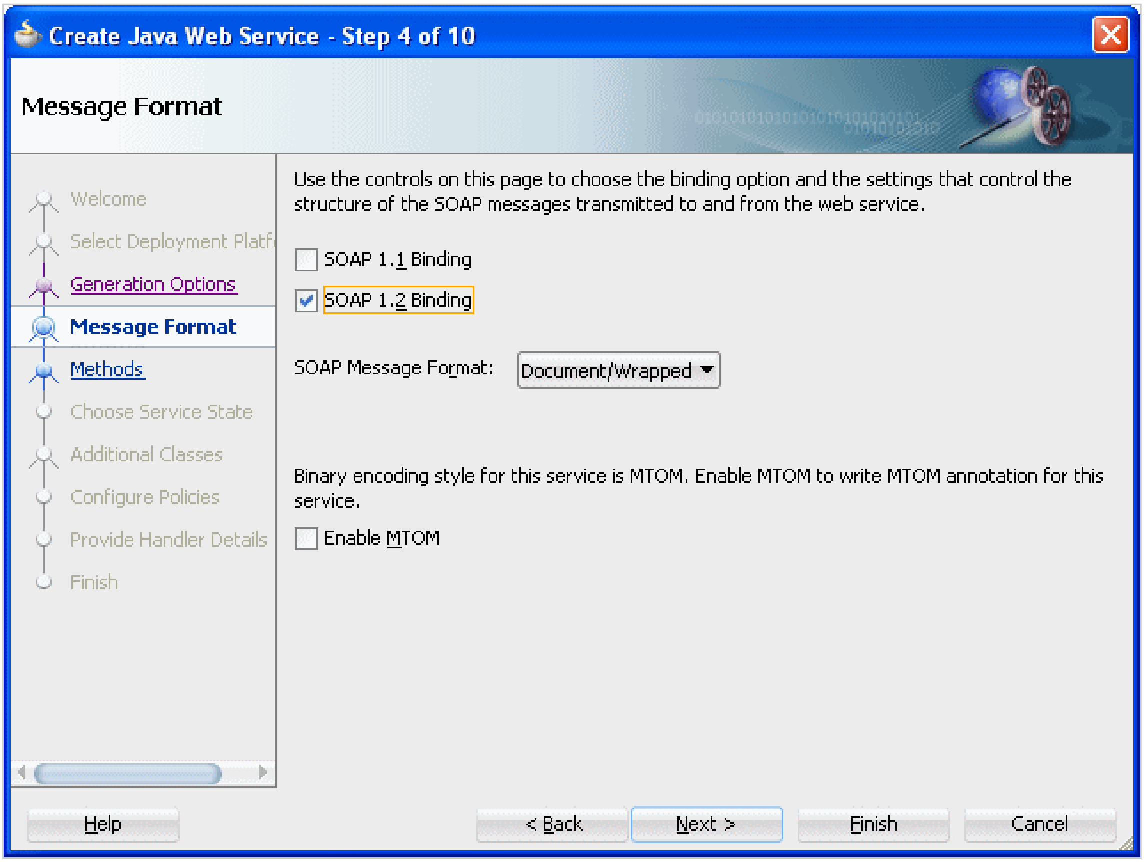
Component To Publish: wizard.HelloService

Web Service Name: MyWebService1

Port Name: MyWebService1Port

☐ Add SEI

5. Click **Next**.
6. In the Message Format step of the wizard, select the **SOAP 1.2 Binding** option.


Create Java Web Service - Step 4 of 10

Message Format

Use the controls on this page to choose the binding option and the settings that control the structure of the SOAP messages transmitted to and from the web service.

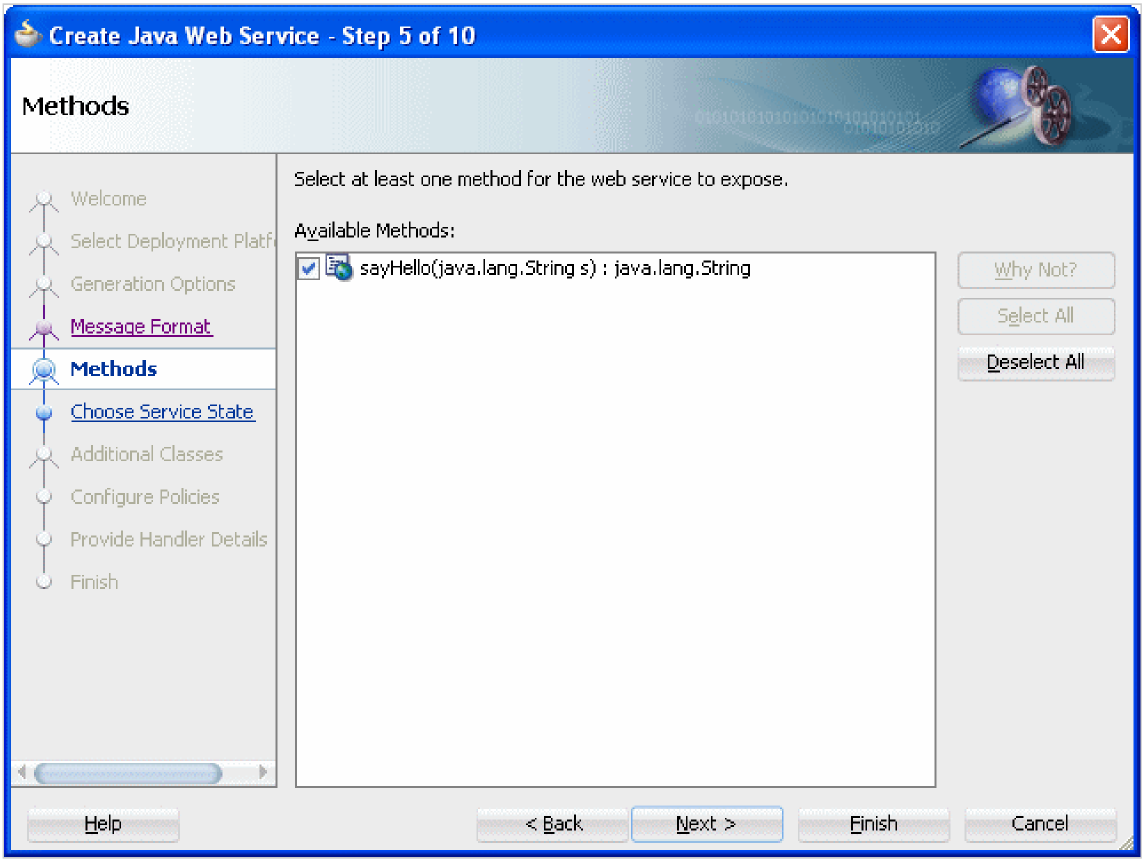
- [ ] SOAP 1.1 Binding
- [✓] SOAP 1.2 Binding

SOAP Message Format: Document/Wrapped

Binary encoding style for this service is MTOM. Enable MTOM to write MTOM annotation for this service.

- [ ] Enable MTOM

7. Click **Next**.
8. In the Methods step of the wizard, all the possible methods to publish are displayed so that you can select the ones you wish to publish. Because in this case there is only one, it is selected by default. The remaining pages are for including any additional classes the service may need, configuring policies for the service, and providing the handler details.
   You will not change any of these values, so click **Finish** at any of these screens to create the web service.


Create Java Web Service - Step 5 of 10

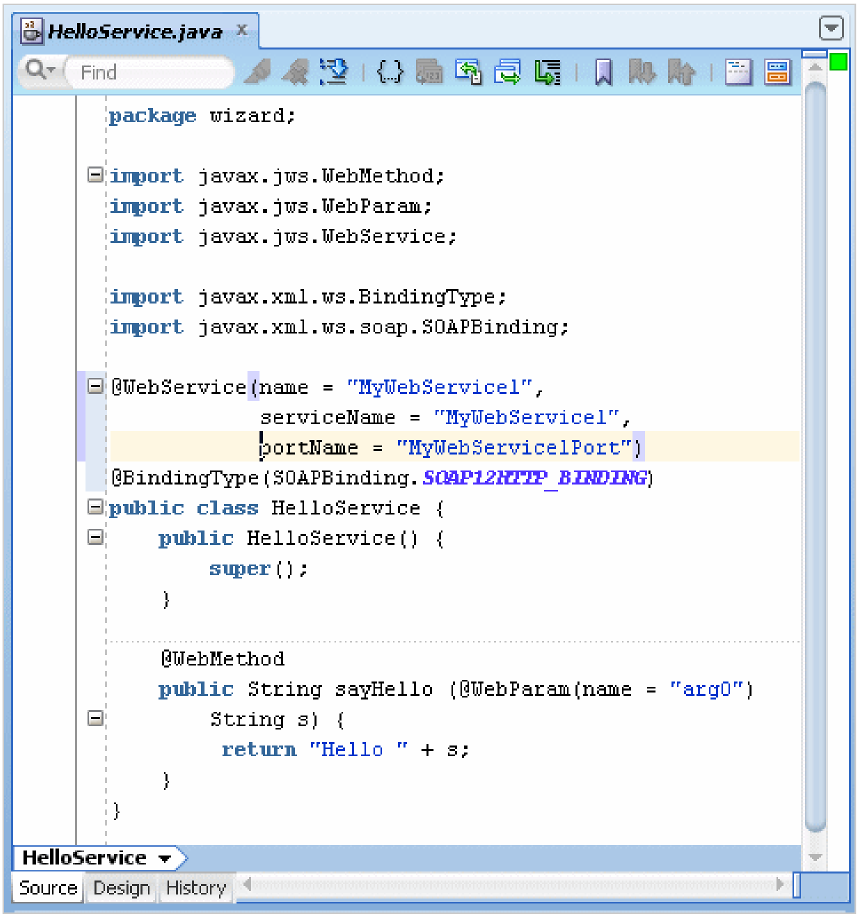
Methods

Select at least one method for the web service to expose.

Available Methods:

- [✓] sayHello(java.lang.String s) : java.lang.String

9. The class definition is updated with the annotation needed to publish the web service. Make sure that the port name is set to **MyWebService1Port**, or change it to this value in the editor.



```
package wizard;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

import javax.xml.ws.BindingType;
import javax.xml.ws.soap.SOAPBinding;

@WebService(name = "MyWebService1",
            serviceName = "MyWebService1",
            portName = "MyWebService1Port")
@BindingType(SOAPBinding.SOAP12HTTP_BINDING)
public class HelloService {
    public HelloService() {
        super();
    }

    @WebMethod
    public String sayHello (@WebParam(name = "arg0")
        String s) {
         return "Hello " + s;
    }
}
```

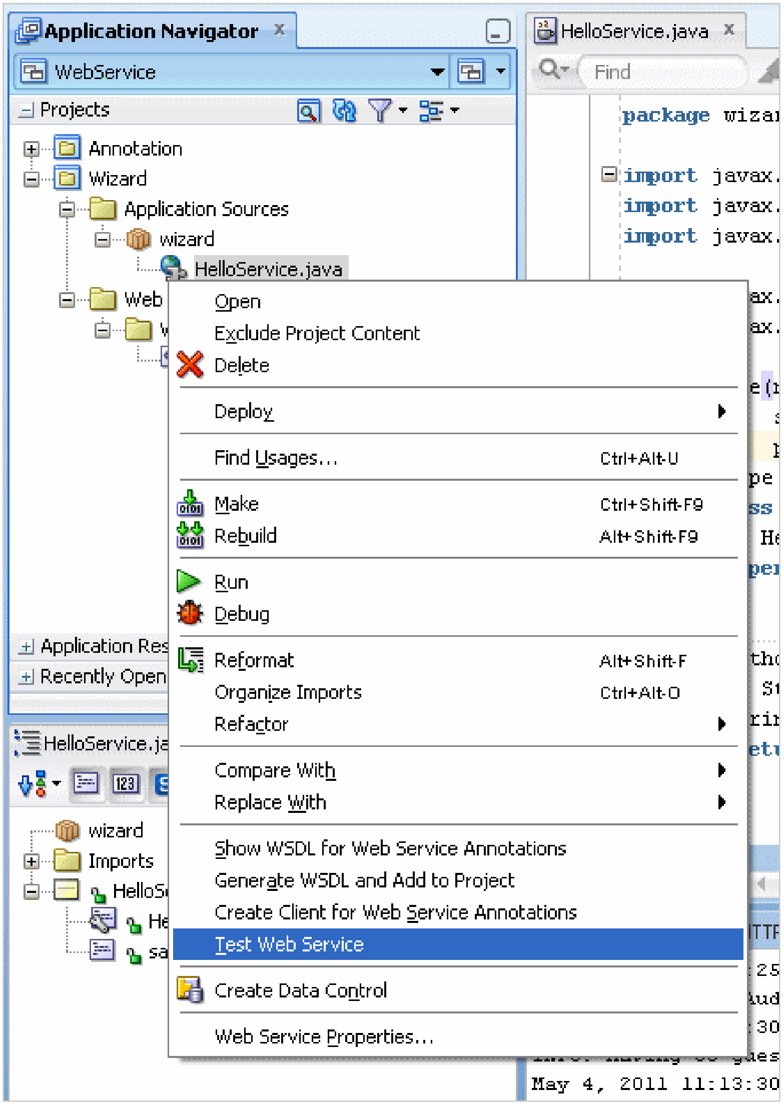
10. Click **Save All** to save your work.

Step 3: Testing the Web Service

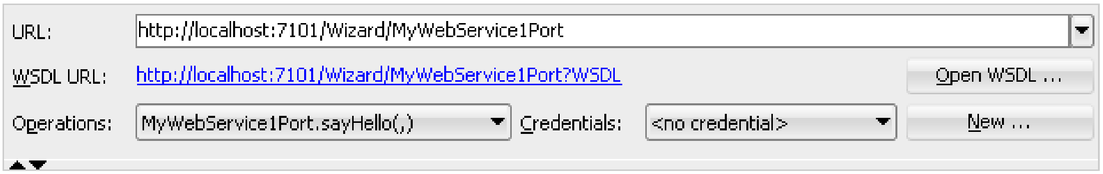
In this section you compile, deploy and test the web service. Just as before, you use the HTTP Analyzer for testing the web service. When you test web services using the analyzer, the service is compiled and deployed to the integrated server automatically. The analyzer is then invoked, enabling you to send values to and receive values from the web service.

1. In the Application Navigator, right-click the **HelloService.java** node in the Wizard project and select **Test Web Service** from the context menu.

   This option invokes the integrated server, deploys the service and then starts the analyzer. It may take a few seconds to start the Integrated server if it is being run for the first time.



Just like earlier, the top portion of the HTTP Analyzer editor displays the URL for the web service, WSDL URL, and exposed Operations.



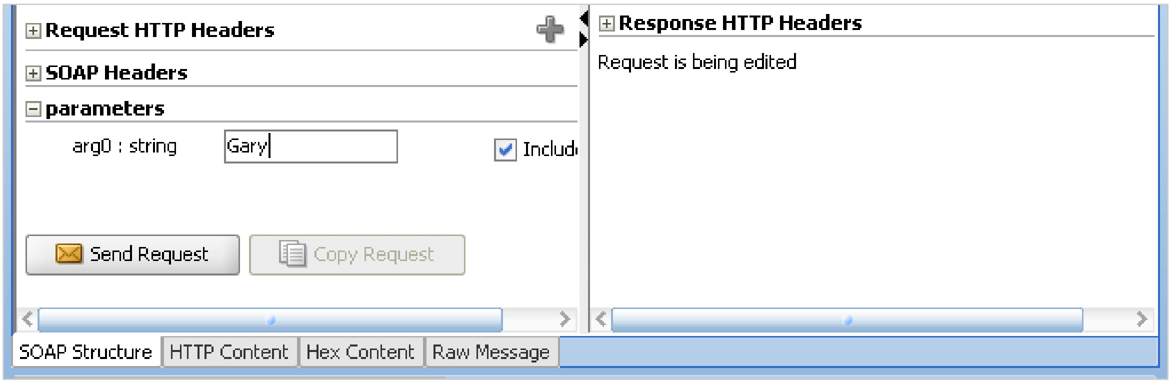
URL: http://localhost:7101/Wizard/MyWebService1Port

WSDL URL: http://localhost:7101/Wizard/MyWebService1Port?WSDL

Operations: MyWebService1Port.sayHello(,)

Credentials: <no credential>

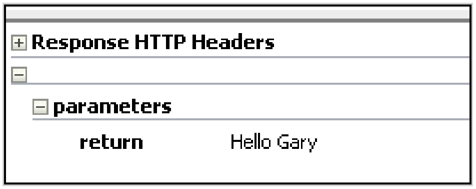
2. In the Request area, enter <your name> in the **arg0** field and click **Send Request**.



arg0 : string Gary

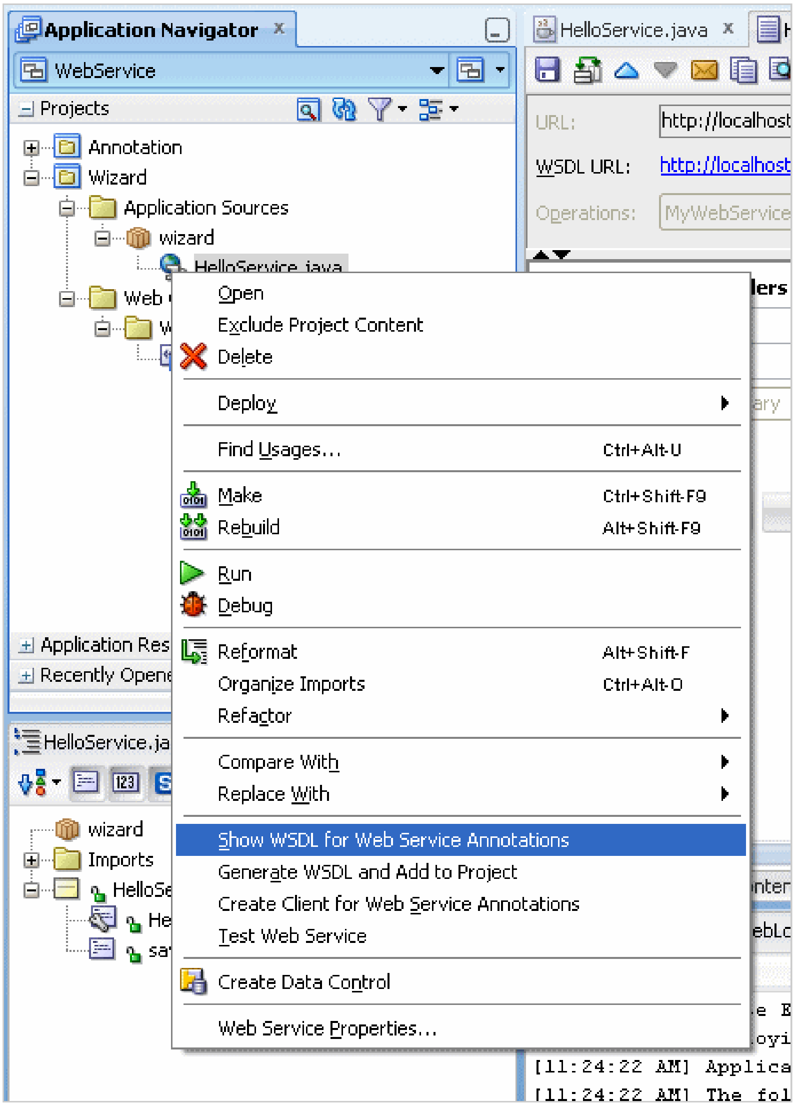
Request is being edited

The analyzer sends the request to the service, and after a few seconds the return parameter is displayed.

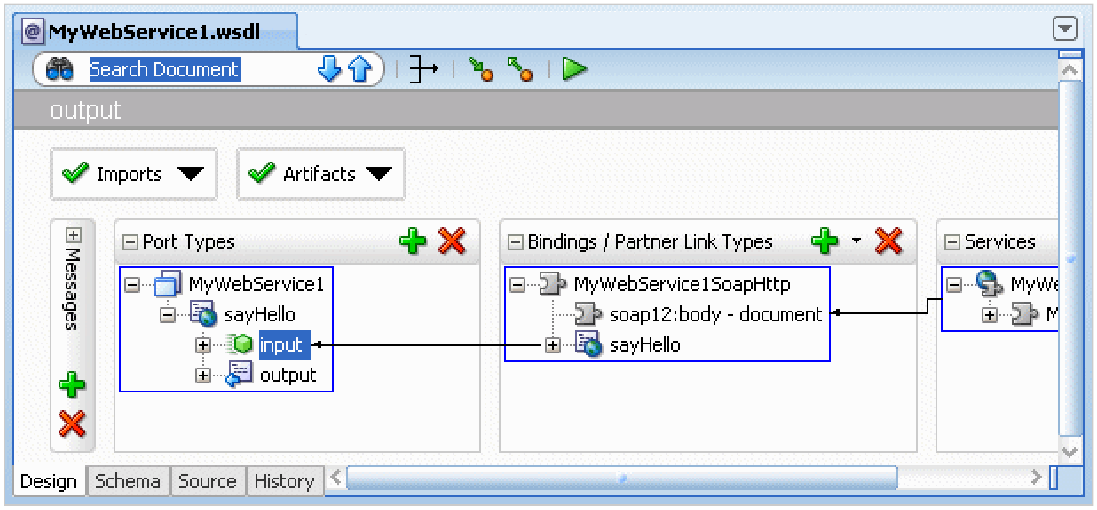


You have now used a wizard in JDeveloper to annotate a Java class into a web service. This is an example of a bottom-to-top approach for creating a web service. You created a java class and then made it into a web service. Sometimes a web service exists that you want to use in some way. That is an example of a top-to-bottom approach. In the next section of this tutorial, you see how to consume an existing web service.

3. In preparation for the next task of creating a web service from a WSDL file, generate the WDSL to a file. Right-click the **HelloService.java** class in the Navigator and select **Show WSDL for Web Service Annotations**.

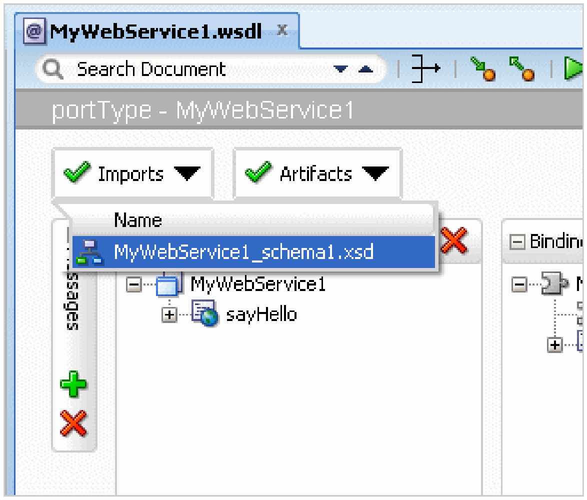


The generation of the WSDL file starts, and then MyWebService1.wsdl displays in the Design editor. You can expand and select the nodes to visualize the flows.

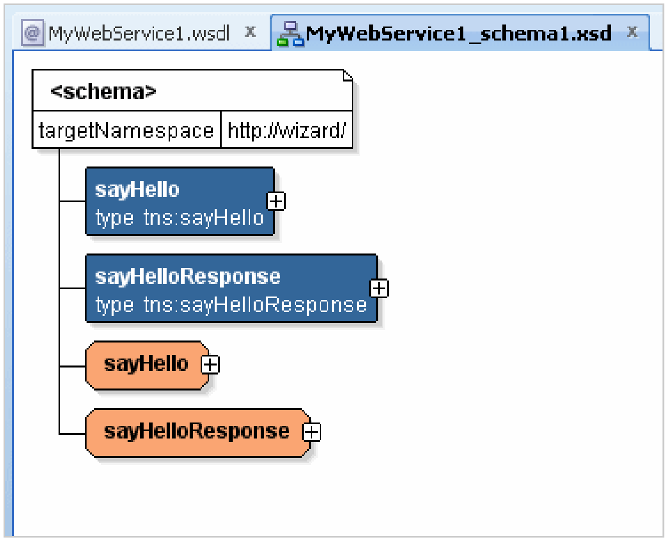


4. Click **Imports** to see what other artifacts the WSDL uses. In this case it uses MyWebService1_schema1.xsd.

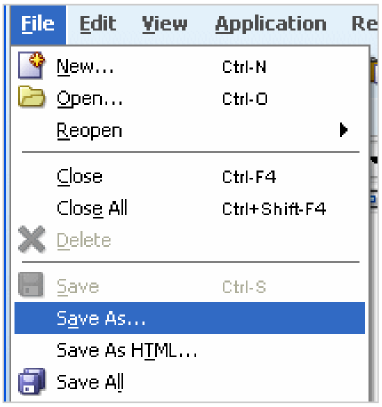
5. Select **MyWebService1_schema1.xsd** from the drop down list.



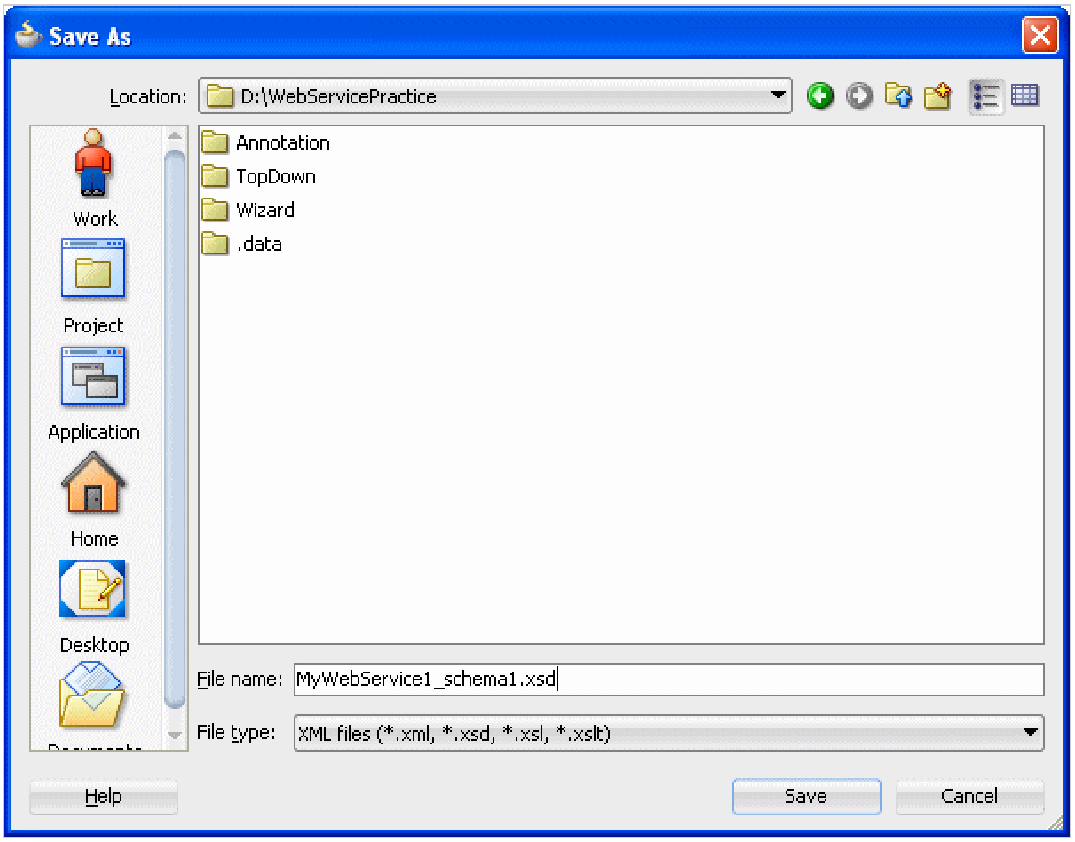
6. The schema document opens in a visual editor window that shows the relationships of the defined elements.



7. From the menu, select **File | Save As...**



and save the file in the directory of your choice.



Location: D:\WebServicePractice

File name: MyWebService1_schema1.xsd

File type: XML files (*.xml, *.xsd, *.xsl, *.xslt)

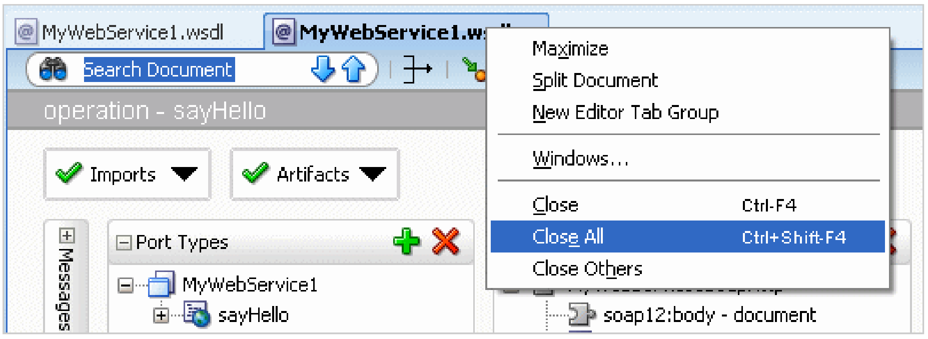
8. Select the **MyWebService1.wsdl** tab and save that file to the same directory you chose for the MyWebService1_schema1.xsd file.

9. Locate the directory where you saved the **MyWebService1.wsdl** file and open it in WordPad or the text editor of your choice.

```
<?xml version="1.0" standalone="yes"?>
<!-- Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is
<definitions
    targetNamespace="http://wizard/"
    name="MyWebService1"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://wizard/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://wizard/"
           schemaLocation="MyWebService1_schema1.xsd"/>
    </xsd:schema>
  </types>
  <message name="sayHello">
    <part name="parameters" element="tns:sayHello"/>
  </message>
  <message name="sayHelloResponse">
    <part name="parameters" element="tns:sayHelloResponse"/>
  </message>
  <portType name="MyWebService1">
    <operation name="sayHello">
      <input message="tns:sayHello"/>
      <output message="tns:sayHelloResponse"/>
    </operation>
  </portType>
  <binding name="MyWebService1PortBinding" type="tns:MyWebService1">
    <soap12:binding transport="http://www.w3.org/2003/05/soap/bindings/HTTP
    <operation name="sayHello">
      <soap12:operation soapAction=""/>
      <input>
        <soap12:body use="literal"/>
      </input>
      <output>
        <soap12:body use="literal"/>
      </output>
```

10. After you examine the file, **close** the text editor window.

11. Close all the tabs in the Editor and collapse the **Wizard** node in the Application Navigator.



Previous 1 **2** 3 4 5 6 Next