NetBeans 5.5 Web Services Consumption in Visual Web Pack Specification

NetBeans 5.5 Web Services Consumption in Visual Web Pack Version 1.0. 08/18/06 - initial version - Sanjay Dhamankar

revised 01/28/07

Note :

See also :

1. "Using NetBeans 5.5 Web Services Client to consume Web Services in Visual Web Pack" for a user documentation.
2. Services Tab Requirement Analysis : http://jupiter.czech.sun.com/wiki/view/Main/ServicesTabRequirements
3. Web Services Core Team http://jupiter.czech.sun.com/wiki/view/Jet/NeteansWebServicesCore
4. Services API : http://xdesign-tools.czech.sun.com:8080/JSPWiki/attach/Service_API/overview.html

Introduction to Web Services

According to the W3C a **Web service** is a software system designed to support interoperable machine-to-machine interaction over a network.

There are two types of web services supported by IDE.

JAX-WS web service clients (Java EE 5). For consuming JAX-WS web services, there is one type of web service client, the IDE-generated static stub. The IDE generates the stub and other artifacts, packages them in the archive, and deploys them. Since JAX-WS does not work with deployment descriptors, but uses annotations within the Java code instead, a J2EE container-generated static stub, which implies the use of deployment descriptors, would be superfluous.

JAX-RPC web service clients (J2EE 1.4). For consuming JAX-RPC web services, there are two types of web service clients:

- J2EE Container-generated static stub. This type is based on JSR-109, which enhances JSR-101 by defining the packaging of web services into standard J2EE modules, including a new deployment descriptor, and defining web services that are implemented as session beans or servlets. This is the recommended and portable (via J2EE 1.4 specification) type. When one chooses this type, the IDE adds deployment information in the deployment descriptors

and the container does the generation of the stub and other artifacts.

- IDE-generated static stub. This type is based on JSR-101, which defines the mapping of WSDL to Java and vice versa. It also defines a client API to invoke a remote web service and a runtime environment on the server to host a web service. This type is not portable. When one chooses this type, the IDE generates the stub and other artifacts, packages them in the archive, and deploys them.

One of the technologies available in J2EE for developing and deploying web services is Java API for XML-Based RPC (JAX-RPC). However, using JAX-RPC technology requires a developer to specify a lot of information in deployment descriptors such as webservices.xml, web.xml, and ejb-jar.xml. These requirements are seen by many programmers as cumbersome and confusing.

To simplify the web services programming model, the Java API for XML Web Services (JAX-WS) was introduced. The main focus of JAX-WS technology is ease of web services development and deployment. A lot of this simplicity results from annotations. "Developing Web Services Using JAX-WS" shows some of these annotations in use. These and other JAX-WS features have eliminated the requirement for deployment descriptors.

Check the FAQ about the differences between the two types of Web Services. For detailed information, refer to the full specifications :

The JSR-109 Specification

The JSR-101 Specification

Since the WSDL file governs the interface to the web service, one may not add new operations to web services that are created from a WSDL file, because these operations will not be reflected back in the WSDL file.

Web Service Clients

A web service client is created to consume (i.e., use) a specific web service. The way in which a web service client consumes a web service depends on the way in which the provider makes the web service available (usually the provider also publishes the API for the web service) :

- The provider distributes a URL to the WSDL file of a running web service.
- The provider distributes a WSDL file, which is available to the user on user's local file system.
- The provider distributes a NetBeans project that defines the web service.

Creating a Web Service Client

A web service can be consumed in a web application creating a Web Service Client.

To create a web service client, the user needs to do the following :

1. Create a web application project. e.g. from the New Project Dialog, choose “Web” from the “Categories” and “Visual Web Application” from “Projects”.
2. To consume web services user needs to create the web service client. Expand the “Web Application” node (from the “Projects” tab). “Web Service Client...” is available under most of the menus which contain “New”. e.g. ["Web Application - > Web Pages -> New -> Web Service Client... "]
3. Access the WSDL file of the web service that the web service client is to consume. Depending on what the provider has distributed, do the following:

- To generate a client from a running web service, type or paste the web service's URL. If the user is behind a corporate firewall, click Proxy Settings and set the proxy host and port number. Click Retrieve WSDL.
- The WSDL file is successfully downloaded when the IDE fills Local Filename field with the name of a WSDL file, such as TravelWS.wsdl.
- To generate a client from a WSDL file on the local file system, click Existing WSDL File and browse to the WSDL file on the local file system.
- Specify a package where the client files are to be generated. This package will need to be imported in users' backing bean so that the Web Service operations are available in the backing bean.
- Select the web service client type from the Client Type drop-down:
- J2EE Container-generated static stub. This type is based on JSR-109, which enhances JSR-101 by defining the packaging of web services into standard J2EE modules, including a new deployment descriptor, and defining web services that are implemented as session beans or servlets. This is the recommended and portable (via J2EE 1.4 specification) type. When user chooses this type, the IDE adds deployment information in the deployment descriptors and the container does the generation of the stub and other artifacts.
- IDE-generated static stub. This type is based on JSR-101, which defines the mapping of WSDL to Java and vice versa. It also defines a client API to invoke a remote web service and a runtime environment on the server to host a web service. This type is not portable. When user chooses this type, the IDE generates the stub and other artifacts, packages them in the archive, and deploys them.

4. Finish.

For JAX-WS web services (Java EE 5 specification), the IDE runs the “wsimport” tool, which does the following:

- Creates a new WSDL file. The abstract portion is copied as is from the original file, while the concrete portion is "normalized" from the original. For example, a SOAP binding element is created if it did not exist in the WSDL file. Also, the service element is created to be compatible with the binding, and receives the web service name that user specified

in the wizard, replacing the service element in the original WSDL file.

- Creates an implementation class. The operations found in the WSDL file are added to this class. In the IDE, an implementation class is identified by the name of the portType element defined in the WSDL together with the suffix "_Impl" (for web applications) or "_Bean" (for EJB modules).

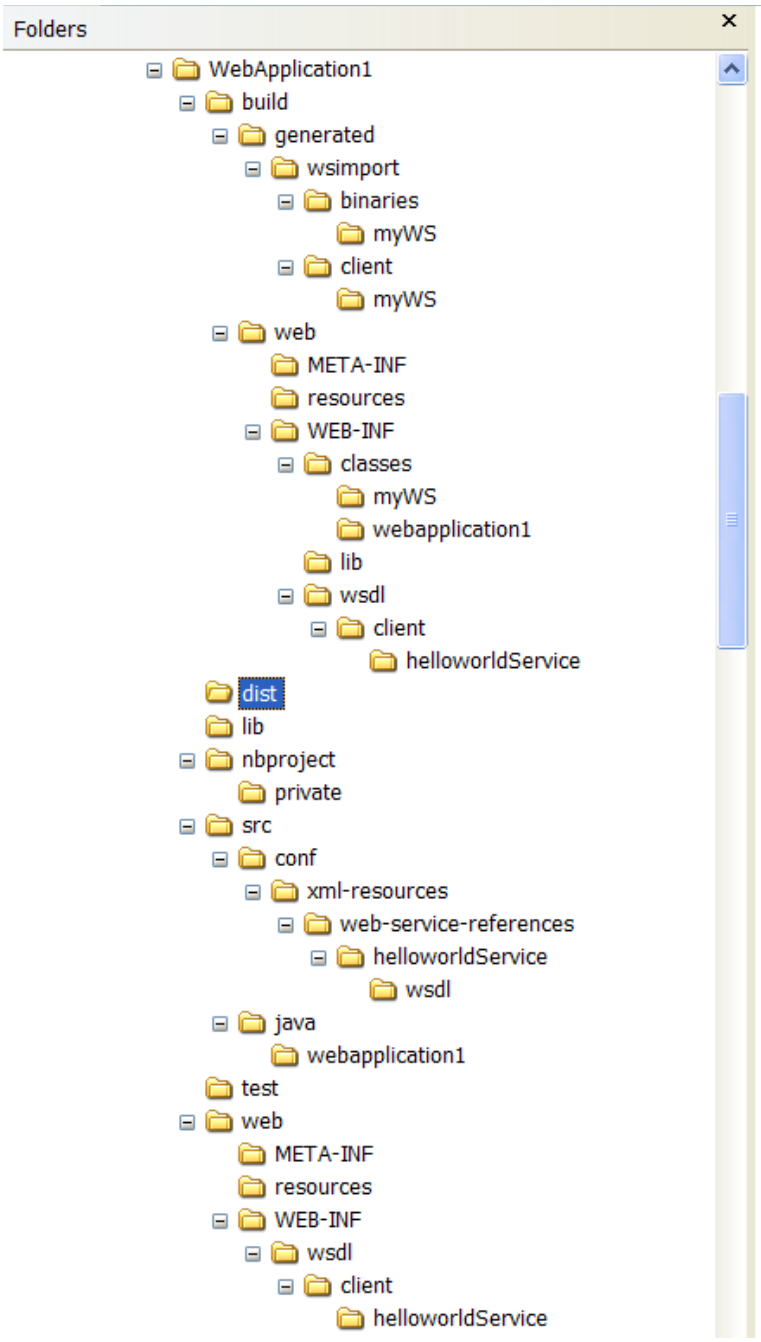
For JAX-RPC web services (J2EE 1.4 specification), the IDE runs the wscompile tool, which does the following:

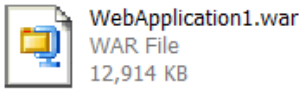
- Creates a new WSDL file. The abstract portion is copied as is from the original file, while the concrete portion is "normalized" from the original. For example, a SOAP binding element is created if it did not exist in the WSDL file. Also, the service element is created to be compatible with the binding, and receives the web service name that user specified in the wizard, replacing the service element in the original WSDL file.
- Creates the Service Endpoint Interface (the "interface"), which declares the web service's operations. In the IDE, the interface class is identified by the name of the portType element defined in the WSDL. The IDE automatically populates the interface with a declaration for each web service operation from the WSDL file.
- Creates the class for the implementation of the interface. The operations found in the WSDL file are added to this class. In the IDE, an implementation class is identified by the name of the portType element defined in the WSDL together with the suffix "_Impl" (for web applications) or "_Bean" (for EJB modules).
- By default, the WSDL file is housed in the WEB-INF/wsdl (or META-INF/wsdl) folder. The interface and the implementation class are housed in a package within user's project's src folder (in the Files window) and user can access them in the Source Packages node (in the Projects window).

The IDE creates a node for the web service under the “Web Service References” node in the Projects window. For each web service operation defined in the WSDL file, the IDE creates a sub-node with the operation's name. The user should be able to double click the operation node and test the functionality of that operation. The IDE also does the following:

1. Creates a folder, if not already present, named WSDL under web/WEB-INF to contain the WSDL file.
2. Adds a service-ref element to the deployment descriptor (the web.xml file) for this service.
3. Adds a “wscompile” target for this service to the Ant build script (the build.xml file).

   Here are the folders and files generated after adding “helloworldService” web service client and calling build. Note that the name of the package is myWS.





The WebApplication1.war contains following:

Name	Type	Modified	Size	Ratio	Packed	Path
Page1.jsp	JSP File	1/28/2007 4:52 PM	885	0%	885	
context.xml	XML Document	1/28/2007 4:52 PM	74	0%	74	meta-inf\
Manifest.mf	MF File	1/28/2007 4:52 PM	106	0%	106	meta-inf\
stylesheet.css	Cascading Style Sheet Docum...	1/28/2007 4:52 PM	626	0%	626	resources\
faces-config.xml	XML Document	1/28/2007 4:52 PM	560	0%	560	web-inf\
managed-beans.xml	XML Document	1/28/2007 4:52 PM	1,205	0%	1,205	web-inf\
navigation.xml	XML Document	1/28/2007 4:52 PM	215	0%	215	web-inf\
web.xml	XML Document	1/28/2007 4:52 PM	3,427	0%	3,427	web-inf\
HelloIF.class	CLASS File	1/28/2007 4:52 PM	635	0%	635	WEB-INF\classes\myWS\
HelloWorld.class	CLASS File	1/28/2007 4:52 PM	1,437	0%	1,437	WEB-INF\classes\myWS\
ObjectFactory.class	CLASS File	1/28/2007 4:52 PM	1,687	0%	1,687	WEB-INF\classes\myWS\
package-info.class	CLASS File	1/28/2007 4:52 PM	217	0%	217	WEB-INF\classes\myWS\
SayHello.class	CLASS File	1/28/2007 4:52 PM	830	0%	830	WEB-INF\classes\myWS\
SayHelloResponse.class	CLASS File	1/28/2007 4:52 PM	875	0%	875	WEB-INF\classes\myWS\
ApplicationBean1.class	CLASS File	1/28/2007 4:52 PM	1,046	0%	1,046	WEB-INF\classes\webapplication1\
Bundle.properties	PROPERTIES File	1/28/2007 4:52 PM	40	0%	40	WEB-INF\classes\webapplication1\
Page1.class	CLASS File	1/28/2007 4:52 PM	3,823	0%	3,823	WEB-INF\classes\webapplication1\
RequestBean1.class	CLASS File	1/28/2007 4:52 PM	1,322	0%	1,322	WEB-INF\classes\webapplication1\
SessionBean1.class	CLASS File	1/28/2007 4:52 PM	1,288	0%	1,288	WEB-INF\classes\webapplication1\
activation.jar	Executable Jar File	1/28/2007 4:52 PM	54,829	0%	54,829	WEB-INF\lib\
appbase.jar	Executable Jar File	1/28/2007 4:52 PM	50,635	0%	50,635	WEB-INF\lib\
commons-beanutils.jar	Executable Jar File	1/28/2007 4:52 PM	118,757	0%	118,757	WEB-INF\lib\
commons-collections.jar	Executable Jar File	1/28/2007 4:52 PM	170,902	0%	170,902	WEB-INF\lib\
commons-digester.jar	Executable Jar File	1/28/2007 4:52 PM	109,131	0%	109,131	WEB-INF\lib\
commons-fileupload.jar	Executable Jar File	1/28/2007 4:52 PM	22,379	0%	22,379	WEB-INF\lib\
commons-logging-1.0.4.jar	Executable Jar File	1/28/2007 4:52 PM	26,202	0%	26,202	WEB-INF\lib\
dataprovider.jar	Executable Jar File	1/28/2007 4:52 PM	228,208	0%	228,208	WEB-INF\lib\
defaulttheme.jar	Executable Jar File	1/28/2007 4:52 PM	202,759	0%	202,759	WEB-INF\lib\
errorhandler.jar	Executable Jar File	1/28/2007 4:52 PM	22,810	0%	22,810	WEB-INF\lib\
FastInfoset.jar	Executable Jar File	1/28/2007 4:52 PM	264,213	0%	264,213	WEB-INF\lib\
http.jar	Executable Jar File	1/28/2007 4:52 PM	76,346	0%	76,346	WEB-INF\lib\
jaxb-api.jar	Executable Jar File	1/28/2007 4:52 PM	73,081	0%	73,081	WEB-INF\lib\
jaxb-impl.jar	Executable Jar File	1/28/2007 4:52 PM	778,939	0%	778,939	WEB-INF\lib\
jaxb-xjc.jar	Executable Jar File	1/28/2007 4:52 PM	2,978,657	0%	2,978,...	WEB-INF\lib\
jaxws-api.jar	Executable Jar File	1/28/2007 4:52 PM	23,618	0%	23,618	WEB-INF\lib\
jaxws-rt.jar	Executable Jar File	1/28/2007 4:52 PM	616,742	0%	616,742	WEB-INF\lib\
jaxws-tools.jar	Executable Jar File	1/28/2007 4:52 PM	489,662	0%	489,662	WEB-INF\lib\
jsf-api.jar	Executable Jar File	1/28/2007 4:52 PM	364,449	0%	364,449	WEB-INF\lib\
jsfcl.jar	Executable Jar File	1/28/2007 4:52 PM	204,803	0%	204,803	WEB-INF\lib\
jsf-impl.jar	Executable Jar File	1/28/2007 4:52 PM	806,317	0%	806,317	WEB-INF\lib\
jsr173_api.jar	Executable Jar File	1/28/2007 4:52 PM	46,047	0%	46,047	WEB-INF\lib\
jsr181-api.jar	Executable Jar File	1/28/2007 4:52 PM	7,995	0%	7,995	WEB-INF\lib\
jsr250-api.jar	Executable Jar File	1/28/2007 4:52 PM	6,165	0%	6,165	WEB-INF\lib\
jstl.jar	Executable Jar File	1/28/2007 4:52 PM	20,682	0%	20,682	WEB-INF\lib\
rowset.jar	Executable Jar File	1/28/2007 4:52 PM	126,268	0%	126,268	WEB-INF\lib\
saaj-api.jar	Executable Jar File	1/28/2007 4:52 PM	18,817	0%	18,817	WEB-INF\lib\
saaj-impl.jar	Executable Jar File	1/28/2007 4:52 PM	274,208	0%	274,208	WEB-INF\lib\
sjsxp.jar	Executable Jar File	1/28/2007 4:52 PM	331,341	0%	331,341	WEB-INF\lib\
sqlx.jar	Executable Jar File	1/28/2007 4:52 PM	150,109	0%	150,109	WEB-INF\lib\
standard.jar	Executable Jar File	1/28/2007 4:52 PM	393,259	0%	393,259	WEB-INF\lib\
webui.jar	Executable Jar File	1/28/2007 4:52 PM	4,134,236	0%	4,134,...	WEB-INF\lib\
helloworldService.wsdl	WSDL File	1/28/2007 4:52 PM	1,929	0%	1,929	WEB-INF\wsdl\client\helloworldService\

Calling a Web Service Operation

Once the web service client is created from WSDL file the user could drag and drop the web service method into a java or jsp file. e.g. Here is the code generated by dragging and dropping the web service method “sayHello” of the web service from the sayHello node on to the java code inside the function buttonAction. In the design view a button was present with its action named buttonAction().

```
try { // Call Web Service Operation
myWS.HelloWorld service = new myWS.HelloWorld();
myWS.HelloIF port = service.getHelloIFPort();
// TODO initialize WS operation arguments here
java.lang.String name = "";
// TODO process result here
java.lang.String result = port.sayHello(name);
System.out.println("Result = "+result);
} catch (Exception ex) {
// TODO handle custom exceptions here
}
```

Now the user has access to the web service client code from the Visual Web Pack backing bean actions. User gets all the features of IDE such as code completion etc. when using the Web Services.

NOTE: Following sections have been marked as "[VisualWebPack_NB6]" meaning these feature are not implemented in shortfin but will be implemented in future releases / updates. For design please refer to "Services Tab Requirement Analysis" mentioned under "see also".

Adding a Method Node to a Web Form by drag and drop [VisualWebPack_NB6]

The user can drag and drop a non-void method on a Web Form. When a method is dropped on a Web Form, a data provider instance for the method is added in the backing bean. Since the data provider is for a method, it needs to be associated with a client instance when it gets created in the backing bean. There are three possible ways for a data provider to be associated with a client instance:

- There is no client instance added in the project yet when a method is dropped. In this case, a client instance will automatically be created and the data provider instance will be associated with it
- There is exact one client instance is in the project. Then the data provider instance will be associated with this client instance.
- There are more than one client instance in the project. A dialog will popup to ask user to select a client instance when a method is dropped.

Once the methods are dropped in the Web Form, they are ready to be bound to the data provider bindable components using either the “Bind to Data...” or “Property Bindings...” dialog from the components.

Binding a Simple Component to a Web Service Method [**VisualWebPack_NB6**]

A simple component displays one value. Examples of simple components include Button, Checkbox, Hyperlink, Text Field, or Static Text. For easiest data binding, use components from the Basic category of the Palette.

After user adds a web service to the IDE and to a page, user can call a web service method by using either of the following techniques:

- Bind a method to a component. Users can use this technique to view the results of the web service method, but they cannot update it. User does not need to write any Java code.
- Call the web service method in user's Java code. User can use this technique to view or update the web service.

The examples in the following procedures describe how to call the same web service method by using both techniques.

To bind a component to a method:

- If the web service is not part of the IDE, add the web service to the IDE.
- Drag and drop the method onto the component.
- Right-click the component and choose Bind to Data.
- In the Bind to Data dialog box, click the Data Provider tab.
- Choose the method's data provider.
- Select the data field to display in the component.

For example, consider the following scenario. User wants to use the getPersons method of the TravelWS web service to display the first person's name in a static text field.

- Create a web service client using the TravelWS.wsdl.
- Drag the Basic > Static Text component to the page.
- Drag the getPersons method onto the static text field.
- Right-click the static text and choose Bind to Data.
- Select name from the data field list.

To call a method in the Java code:

If the web service is not part of the IDE, create the web service client.

For example, consider the following scenario. User wants to use the getPersons method of the TravelWS web service to display the first person's name in a static text field.

- Drag the Basic > Static Text component to the Visual Designer.
- Go to the Page1.java source code by selecting the Java tab.
- In the prerender() method, drag and drop the getPersons() method node. It will generate the following code segment.

```
try { // Call Web Service Operation
travelWS.TravelService port = service.getTravelServicePort();
// TODO process result here
java.util.List<travelWS.PersonDTO> result = port.getPersons();
System.out.println("Result = "+result);
} catch (Exception ex) {
// TODO handle custom exceptions here
}
```

- Modify the code by replacing the "System.out.println ..." by `staticText1.setValue( result[0].getName() );`
- Also add the following to the catch block.

```
log("Page1 Initialization Failure", e);
throw e instanceof FacesException ? (FacesException) e: new
FacesException(e);
```

Binding a List Component to a Web Service Method [**VisualWebPack_NB6**]

When user binds a list component to a web service, user creates a connection between them. List components display one or more values at a time.

To bind a list component to a method:

- Drag and drop the method onto the component.
- Right-click the component and choose Bind to Data.
- In the Bind to Data dialog box, click the Data Provider tab.
- Choose the method's data provider.

- Select the value field and display field for the component.

For example, consider the following scenario. User wants to use the getPersons method of the TravelWS to display names in a drop-down list.

- Create a Web Service Client using TravelWS.wsdl.
- Drag the Basic > Dropdown List component to the page.
- Drag the getPersons method onto the drop-down list.
- Right-click the drop-down list and choose Bind to Data.
- Select personId for the value field and name for the display field.

To call a method in Java code:

For example, consider the following scenario. User wants to use the getPersons method of the TravelWS web service to display names in a drop-down list.

- Create a Web Service Client using TravelWS.wsdl.
- Drag the Basic > Dropdown List component to the Visual Designer.
- Drag the getPersons method of TravelWS to the Visual Designer.
- In the Projects window, add a property named personOptions with the type ArrayList.
- In the Projects window, expand project-name > Source Packages > web application-name.
- Right-click Page1.java and choose Add > Property.
- In the New Property Pattern dialog box, enter personOptions in the Name field and enter ArrayList in the Type drop-down list.
- Go to the Page1.java source code by selecting the Java tab.
- In the init() method, enter the following code:

```
personOptions = new ArrayList();
try {
PersonDTO[] persons = travelWSClient1.getPersons();
for( int i = 0; i < persons.length; i ++ )
personOptions.add( new Option(new
Integer(persons[i].getPersonId()), persons[i].getName() ) );
}catch (Exception e) {
log("Page1 Initialization Failure", e);
throw e instanceof FacesException ? (FacesException) e: new
FacesException(e);
}
```

- Right-click in the Java Editor and choose Fix Imports.

- In the Fix Imports dialog box, choose com.sun.rave.web.ui.model.Option.
- Bind the new property to the drop-down list.
- In the Visual Designer, right-click the drop-down list and choose Property Bindings.
- In the right list in the Property Bindings dialog box, select Page 1 > personOptions.
- Click Apply, then click Close.

Binding a Table Component to a Web Service Method [**VisualWebPack_NB6**]

To bind a Table component to a method:

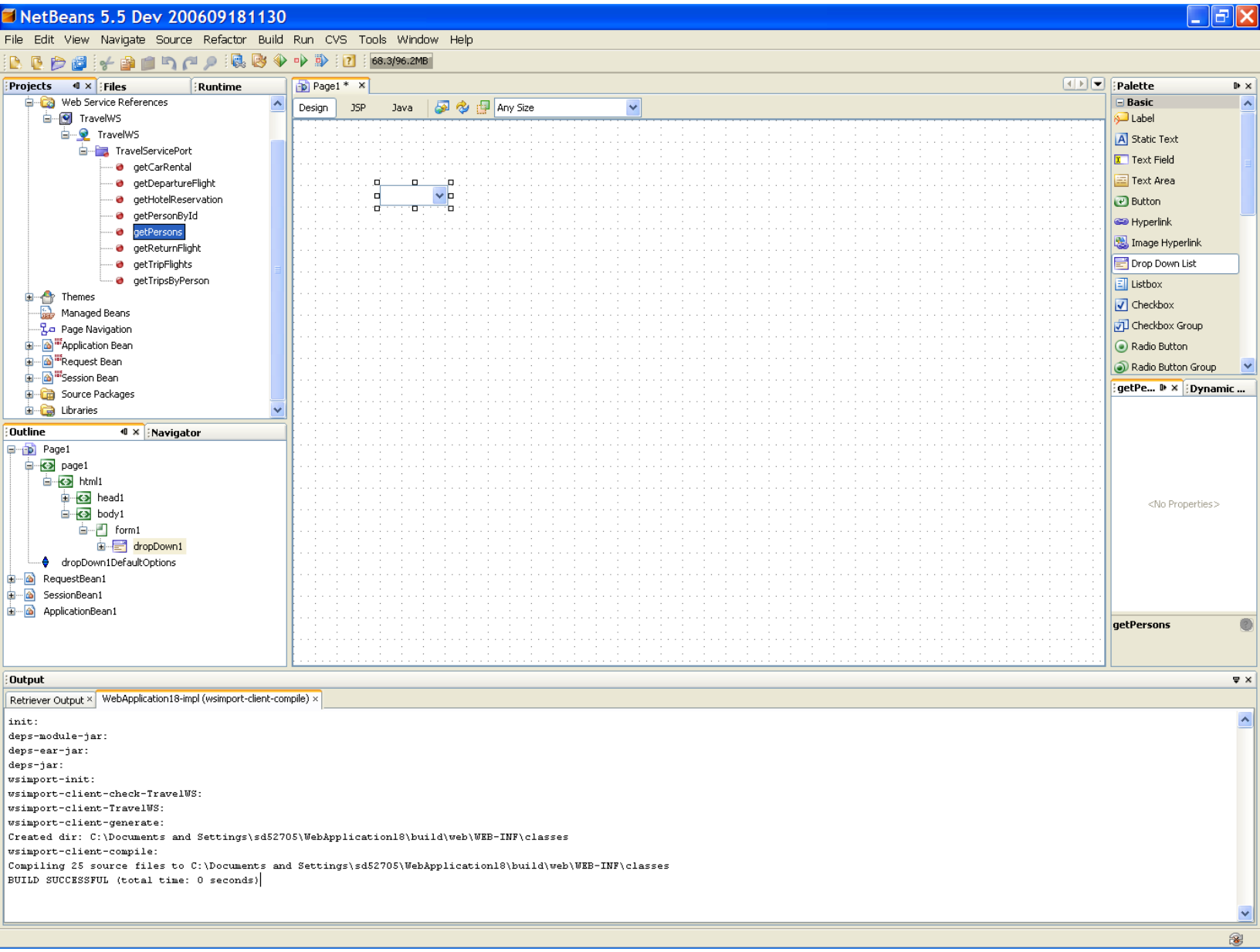
- Create a Web Service Client using TravelWS.wsdl.
- Add the Table component to the Visual Designer.
- Drag and drop the method onto the table.
- For example, consider the following scenario. User wants to use the getPersons method of the TravelWS to display names in a table.
- Drag the Basic > Table component to the page.
- Drag the getPersons method onto the table.

Binding the Drop Down List to a Web Service Method [**VisualWebPack_NB6**]

User could drag and drop a web service method such as “getPersons()” to Drop Down List. When user deploys the application, the Drop Down List displays a list of traveler names.

1. Open the “Projects” window and expand Web Service References > TravelWS > TravelWS > Travel Service Port. The following figure shows the TravelWS web service methods.
2. Drag the `getPersons` method from the Projects window and drop it on the Drop Down List.

   The value in the Drop Down List changes from item 1 to abc. The `"abc"` text indicates that the display field is bound to a `String` object.



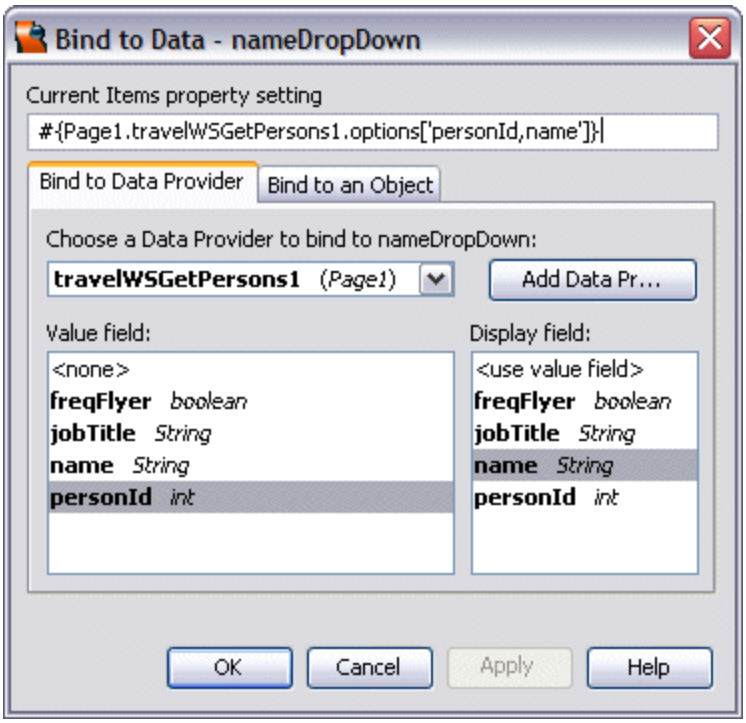
3. Right-click the Drop Down List and choose Bind to Data.

   The Bind to Data dialog box opens.

4. In the Bind to Data Provider tab, set the following three values, as shown in Figure 5.

   Drop-down list: travelWSGetPersons1 (Page 1)
   Value field: personId
   Display field: name

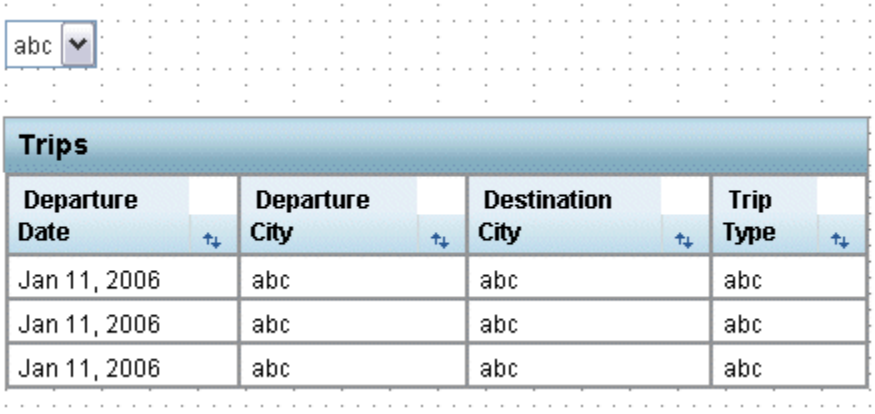
5. Click OK.



Binding the Table to the Web Service Method [**VisualWebPack_NB6**]

This section describes how to bind the Table component to the `getTripsByPerson` method of the TravelWS web service. When the application is deployed it displays master-detail data from a database. When user selects a person from the Drop Down List, the application displays the trip records for that person in the table.

The following figure shows the layout of the table user designs in the next steps.



1. From the Projects window, drag the Web Service References > TravelWS > TravelWS > Travel Service Port > `getTripsByPerson` method and drop it on the Table in the Visual Designer.

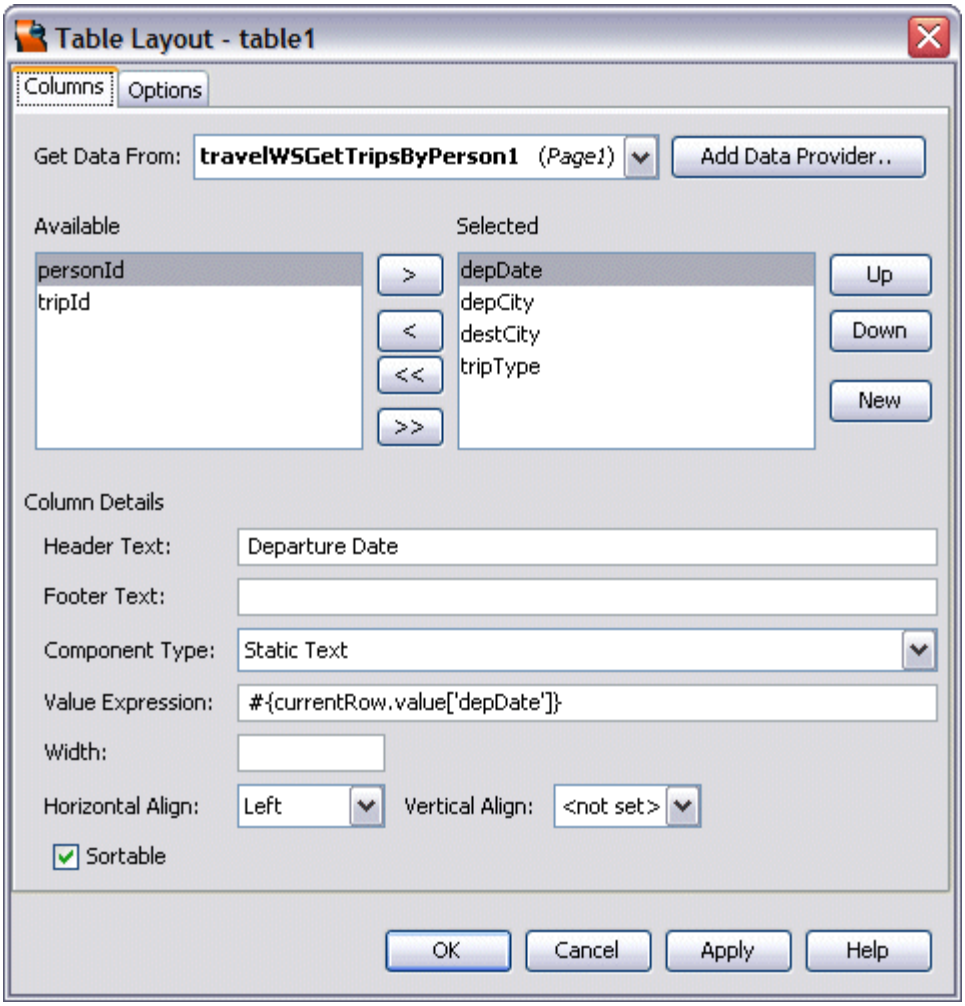
   The value fields of the TravelWS data provider appear as the column names in the Table.

   Note: If the `Choose Target` dialog box opens, choose `table1` and click OK.

1. Right-click anywhere in the Table and choose Table Layout from the pop-up menu.
2. In the Table Layout dialog box, move tripid and personid from the Selected List to the Available List by selecting each one and clicking the < button.
3. Move depDate to the top of the Selected list by selecting it and clicking the `Up` button.
4. Change the Header Text for depDate to `Departure Date`.

   Note that user could drop a converter on the depDate column in the table and set it to show the date only. For more information, see the Using Converters tutorial.

   The following figure shows the Table Layout dialog box with the changes user made so far.



5. Select depCity and change the Header Text to `Departure City`.
6. Select destCity and change the Header Text to `Destination City`.
7. Select tripType and change the Header Text to `Trip Type`.
8. Click the Options tab and change the Title to `Trips`.
9. Click OK to apply the changes and close the Table Layout dialog box.

Populating the Table from the Drop Down List [**VisualWebPack_NB6**]

1. In the Visual Designer, double-click the Drop Down List.

   The Java Editor opens with the insertion point in the `nameDropDown_processValueChange` method.
2. Open the Code Clips tab of the Palette and scroll to the Database and Web Services node.

3. Drag `TravelWS: DropDown Process Value Change` from the Code Clips Palette and drop it in the `nameDropDown_processValueChange` method.

   This code clip gets the selected person id from the Drop Down List, and then sets the id to the data provider for the Table.

4. Drag the `TravelWS: Get the First Person ID` code clip from the Code Clips Palette and drop it right before the end closing bracket in the Java Editor. Note that the statement that includes the `Rowkey` class contains a syntax error because the file does not yet include an import statement for that class. User has to add the import statement in the next step.

   The `Get the First Person ID` code gets the id of the first person from the travelWSGetPerson1 data provider.

5. Right-click anywhere in the Java Editor and choose Fix Imports.

   Fix Imports automatically adds import statements that are needed by user's code and removes unused import statements. Fix Imports does not remove any fully qualified class names user may have in user's code.

6. Scroll up to the `init` method and then drag the `TravelWS: Select the First Person` code clip from the Code Clips Palette and drop it after the comment `Creator-managed Component Initialization`. Enter Control-Shift-F to automatically reformat the code. The result is similar to the following code sample. The `Select the First Person` code is shown in **bold**.

Code Sample 1: `init` Method With Select the First Person code

```
public void init() {
        // Perform initialization inherited from our superclass
        super.init();
        // Perform application initialization that must complete
        // *before* managed components are initialized
        // TODO - add your own initialization code here

        Creator-managed Component Initialization/

        // Perform application initialization that must complete
        // *after* managed components are initialized
        // TODO - add your own initialization code here
        // Initialize the drop down to initialize the first person
       // and the data provider to get the trips for the first person
       Integer firstPerson = getFirstPersonId();
       nameDropDown.setSelected( firstPerson );
       travelWSGetTripsByPerson1.setPersonId( firstPerson );
}
```

This code clip initializes the Drop Down List with the first name from the travelWSGetPersons1 data provider. The code also retrieves the trips for the first person from the travelWSGetTripsByPerson1 data provider. When a different person is selected, the contents of the Trips table change as well.