Using NetBeans 5.5 Web Services Client to consume Web Services in Visual Web Pack

– Sanjay Dhamankar
– Initail version: 08-31-06
– Revised : 01-28-07

For detailed functional specifications refer to "NetBeans 5.5 Web Services Consumption in Visual Web Pack Specification".

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.

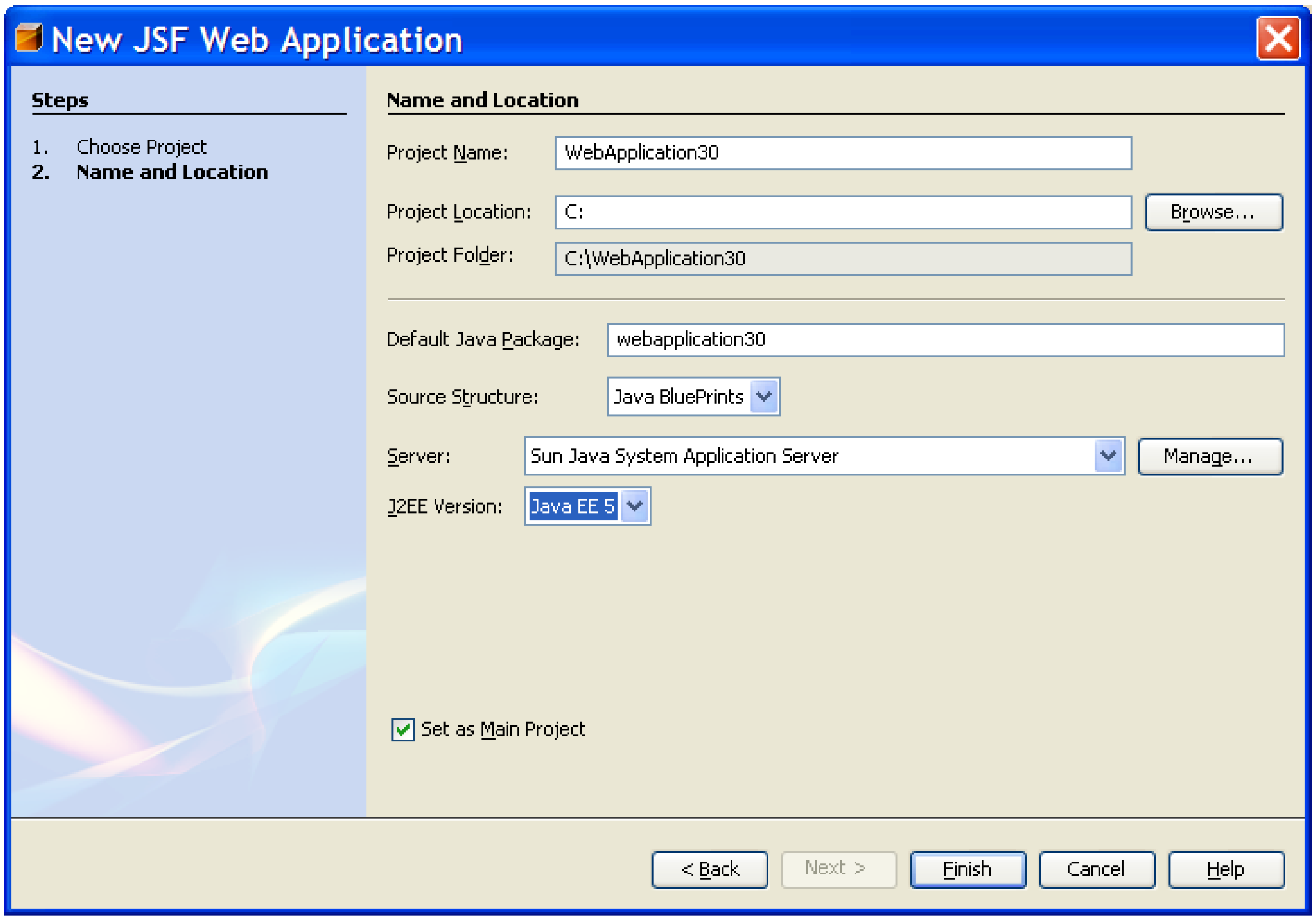
JAX-RPC web service clients (J2EE 1.4). For consuming JAX-RPC web services, there are two types of web service clients:

- 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.

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. Click here for creating web services using Netbeans 5.5 as this document will describe only web service consumption.

Consume the web service

1. Either open an existing or create a new Visual Web Application project. If you are creating a new JSF Application, the dialog may look as follows:


New JSF Web Application

Steps

1. Choose Project
2. **Name and Location**

Name and Location

Project Name: WebApplication30

Project Location: C:

Project Folder: C:\WebApplication30

Default Java Package: webapplication30

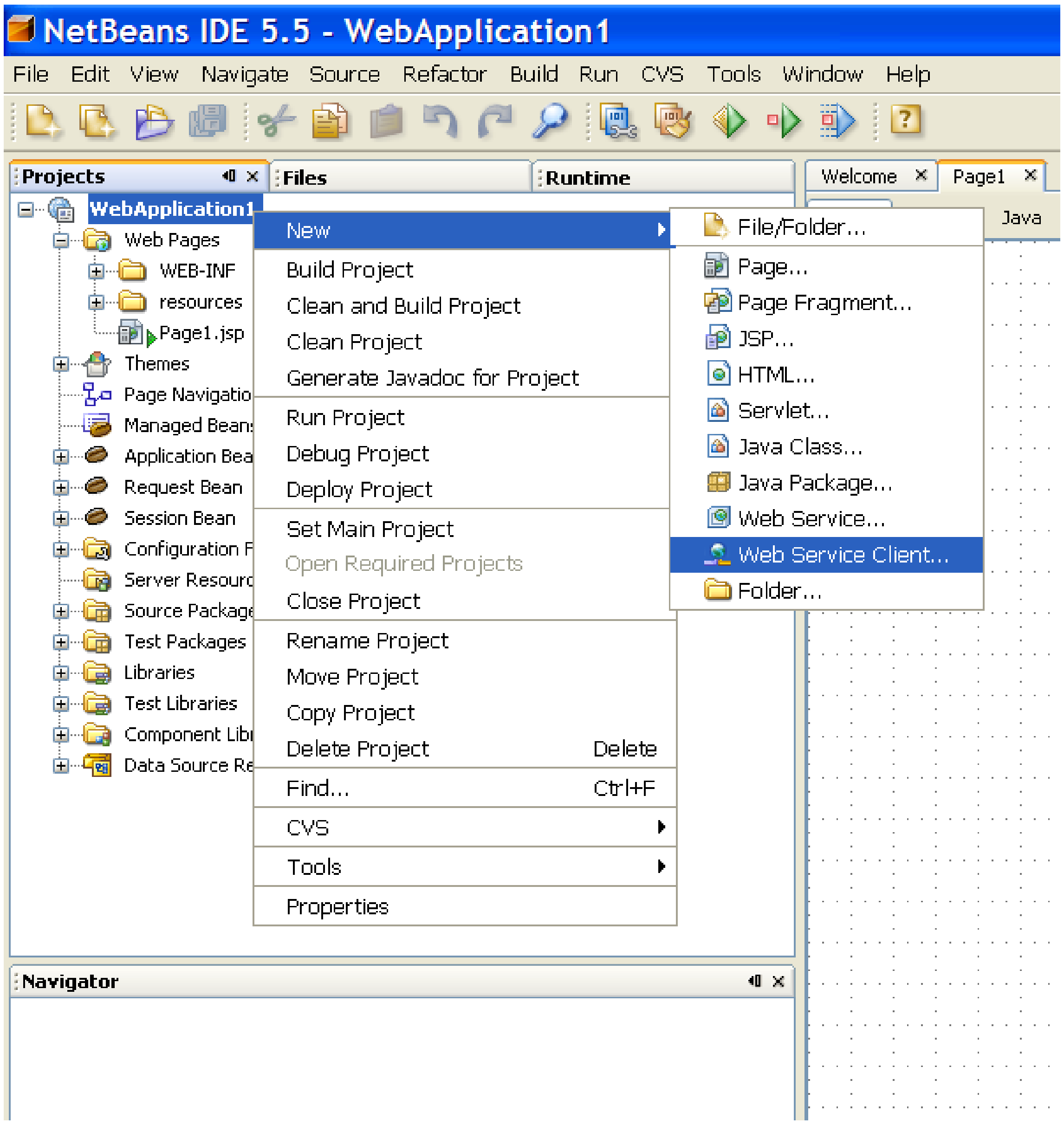
Source Structure: Java BluePrints

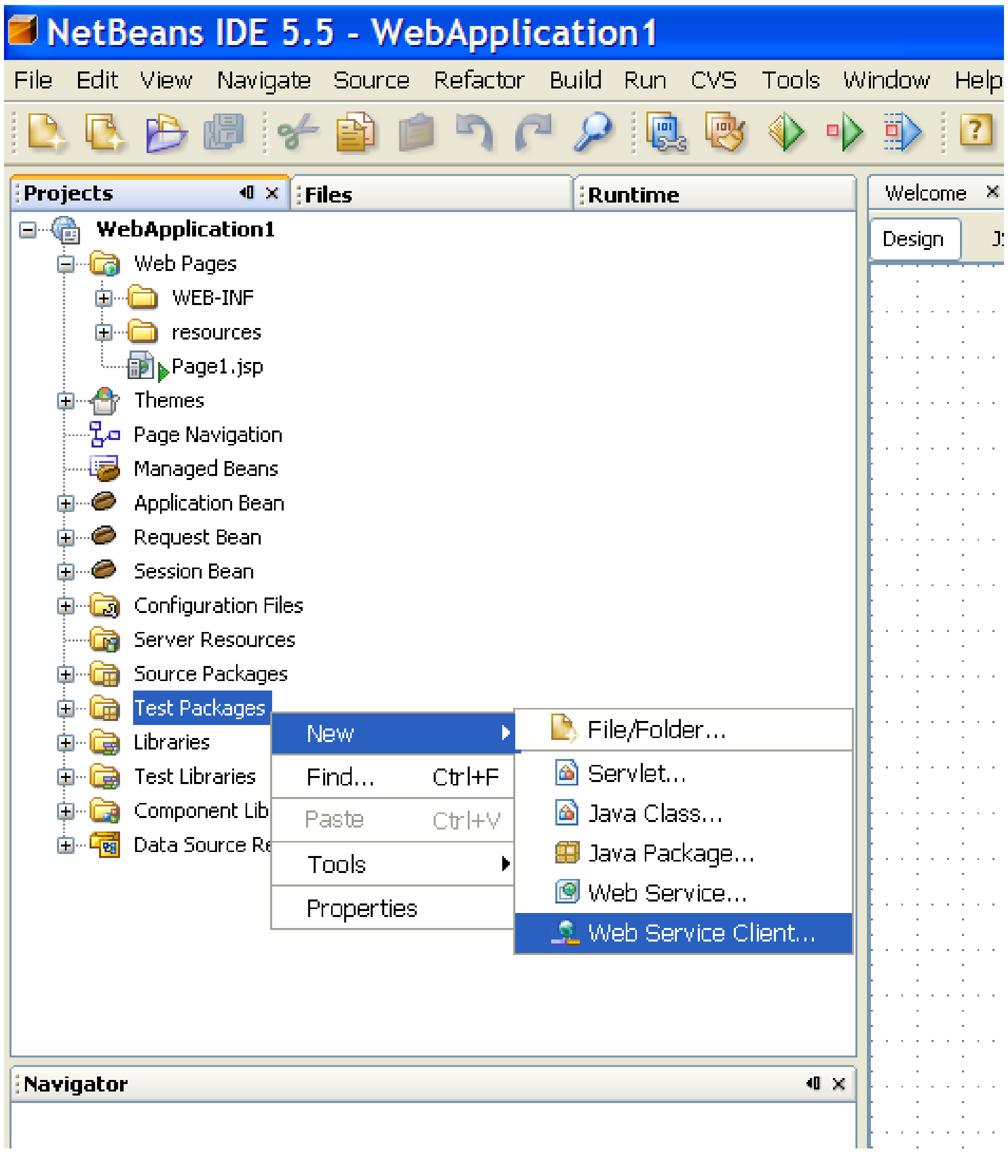
Server: Sun Java System Application Server

J2EE Version: Java EE 5

✓ Set as Main Project

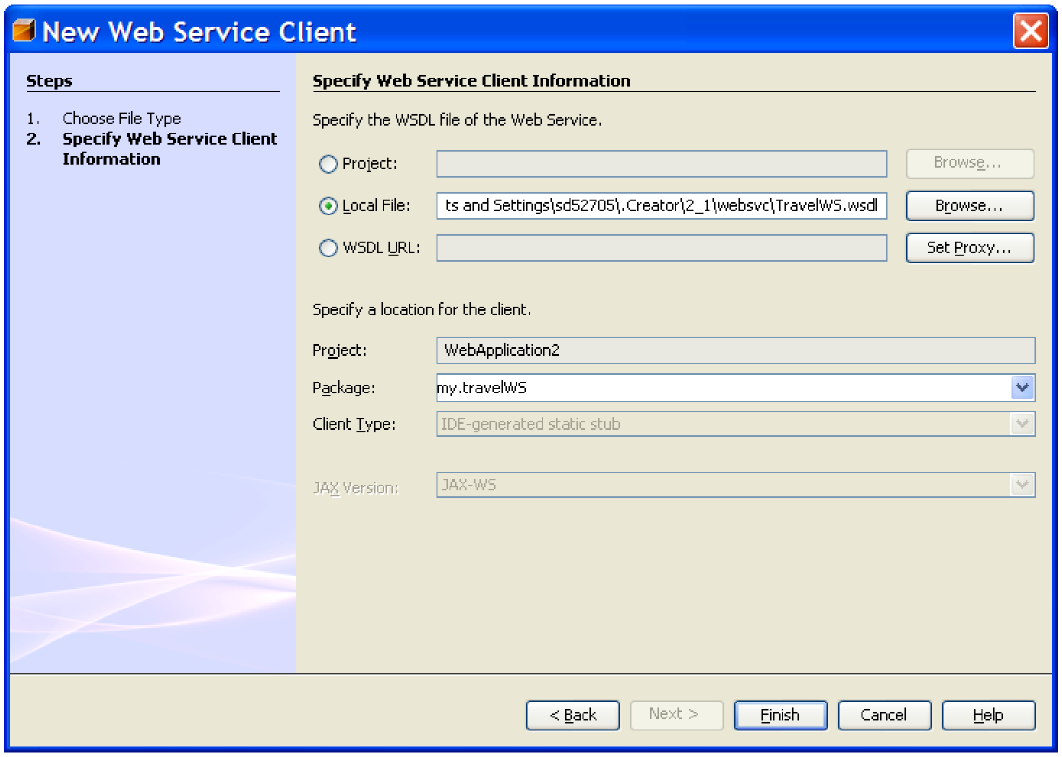
2. To consume web services you need to create the web service client. Expand the "Web Application" node (from the "Projects" tab). "Web Service Client..." is available under most of the menus which contain "New". Here are two of the many menus where the "Web Service Client..." menu is available.


NetBeans IDE 5.5 - WebApplication1
File Edit View Navigate Source Refactor Build Run CVS Tools Window Help
Projects
Files
Runtime
Welcome
Page1
Java
WebApplication1
Web Pages
WEB-INF
resources
Page1.jsp
Themes
Page Navigatio
Managed Bean:
Application Bea
Request Bean
Session Bean
Configuration F
Server Resourc
Source Package
Test Packages
Libraries
Test Libraries
Component Libr
Data Source Re
New
Build Project
Clean and Build Project
Clean Project
Generate Javadoc for Project
Run Project
Debug Project
Deploy Project
Set Main Project
Open Required Projects
Close Project
Rename Project
Move Project
Copy Project
Delete Project Delete
Find... Ctrl+F
CVS
Tools
Properties
File/Folder...
Page...
Page Fragment...
JSP...
HTML...
Servlet...
Java Class...
Java Package...
Web Service...
Web Service Client...
Folder...
Navigator



This depends on the deployment server chosen at the time of project creation.

If Sun App Server is chosen then the JAX-WS client is shown.


New Web Service Client

Steps

1. Choose File Type
2. **Specify Web Service Client Information**

Specify Web Service Client Information

Specify the WSDL file of the Web Service.

- Project:
- [✓ Local File:] ts and Settings\sd52705\.Creator\2_1\websvc\TravelWS.wsdl
- WSDL URL:

Specify a location for the client.

Project: WebApplication2
Package: my.travelWS
Client Type: IDE-generated static stub

JAX Version: JAX-WS

If TomCat is chosen then the JAX-RPC client is shown.

2. Specify either the Net Beans project where the web service is created, or a local WSDL file or URL of the WSDL file. For URL, specify the proxy settings if you are behind the firewall. Typically, an error such as the following is displayed in the Web Service Client wizard when the proxy settings for retrieving a WSDL file have not been set correctly:

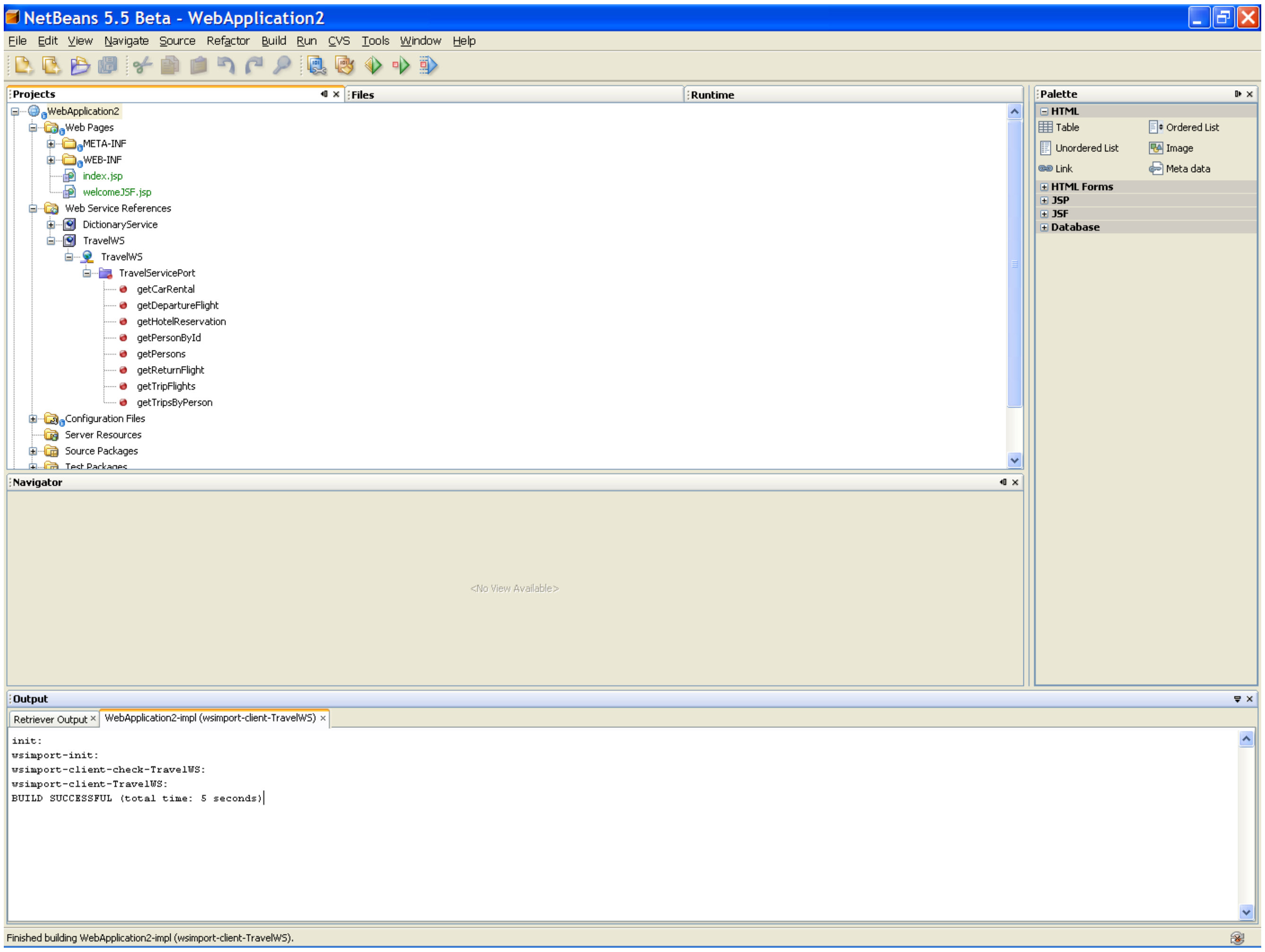
Download failed. I/O exception: (Check the proxy settings.)

Do the following to check and set the proxy:

- Click Proxy Settings in the Web Service Client wizard.
- In the HTTP Proxy Settings window, set the proxy host and port number.
- The changes take effect when you click OK.

3. Specify the location for the client. Leave the project as default, specify the package name (remember this as we might need to import this in the backing bean), Client type (default for JAX-WS) J2EE Container-generated static stub (for JAX-RPC) (see above)

4. Click Finish.



```
init:
wsimport-init:
wsimport-client-check-TravelWS:
wsimport-client-TravelWS:
BUILD SUCCESSFUL (total time: 5 seconds)
```

Finished building WebApplication2-impl (wsimport-client-TravelWS).

5. Import the package specified in the dialog above in the backing bean (SessionsBean1.java) by manually typing the import statement (in future this manual procedure may be eliminated). The statement looks like “import my.travelWS.TravelService;”. Note that here the package name is “my.travelWS”. See the code in the red ellipse in the diagram below. Here is sample code which will be generated when the user drags and drops the web service method “**getCarRental**” in the “.java” file. (e.g. Page1.java) T**he user still needs to type the “import” statement by hand.** Please note that the // TODO part needs to reflect what the user intends to do with these web service methods and must be modified manually. :

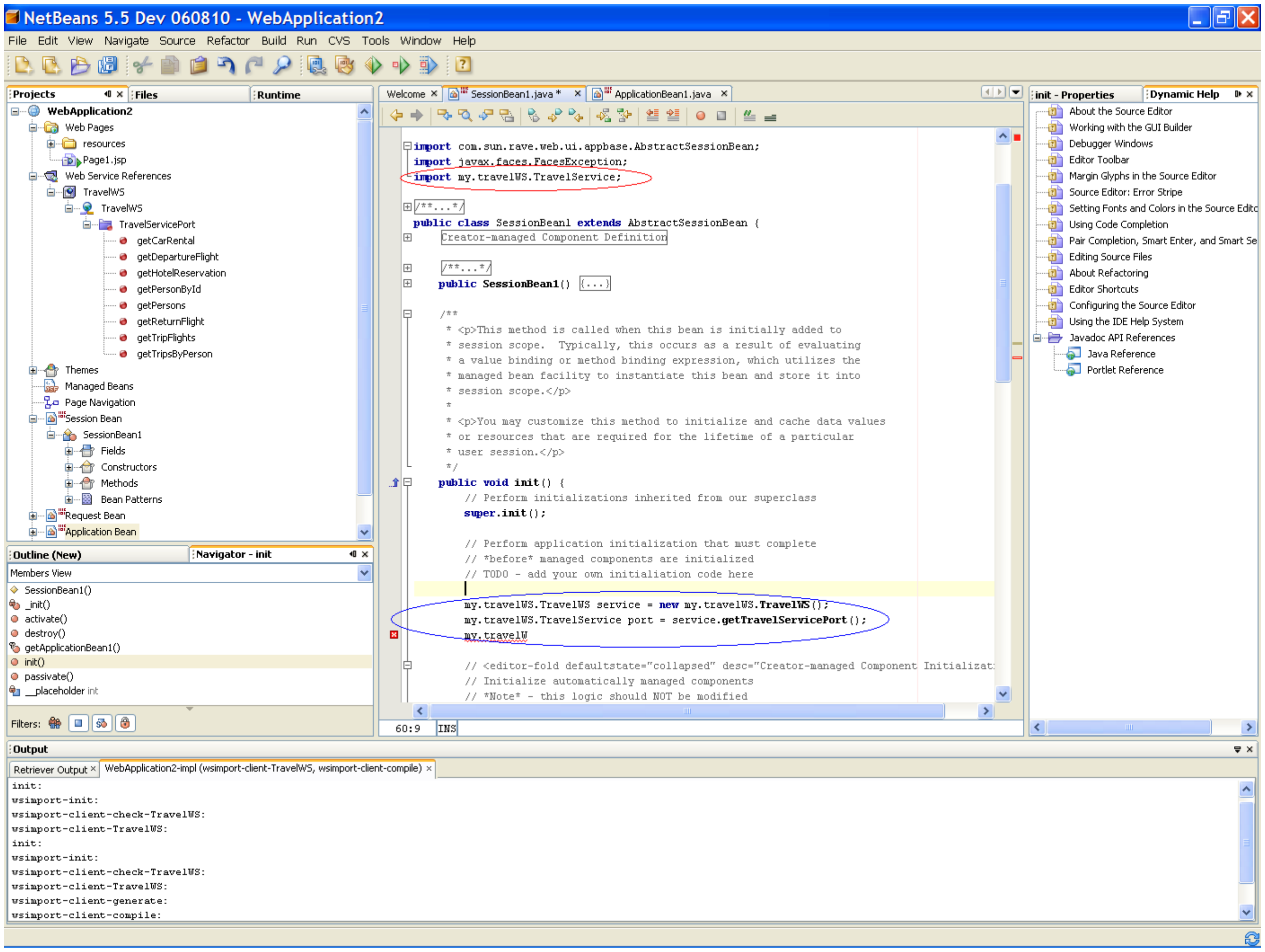
```
try {
        my.travelWS.TravelWS service = new my.travelWS.TravelWS();
        my.travelWS.TravelService port = service.getTravelServicePort();

         // TODO initialize WS operation arguments here
        java.lang.Integer tripId = null;

        // TODO process result here
        my.travelWS.CarRentalDTO result = port.getCarRental(tripId);
} catch (Exception ex) {
        // TODO handle custom exceptions here

}
```

6. Now you have access to the web service client code from the Visual Web Pack backing bean actions. Look at the code in the blue ellipse. Here's an example with the TravelWS.wsdl shipped with Visual Web Pack. You get all the features of IDE such as code completion etc. when you are using the Web Services.



```
import com.sun.rave.web.ui.appbase.AbstractSessionBean;
import javax.faces.FacesException;
import my.travelWS.TravelService;

/**...*/
public class SessionBean1 extends AbstractSessionBean {
    Creator-managed Component Definition

    /**...*/
    public SessionBean1() {...}

    /**
     * <p>This method is called when this bean is initially added to
     * session scope.  Typically, this occurs as a result of evaluating
     * a value binding or method binding expression, which utilizes the
     * managed bean facility to instantiate this bean and store it into
     * session scope.</p>
     *
     * <p>You may customize this method to initialize and cache data values
     * or resources that are required for the lifetime of a particular
     * user session.</p>
     */
    public void init() {
        // Perform initializations inherited from our superclass
        super.init();

        // Perform application initialization that must complete
        // *before* managed components are initialized
        // TODO - add your own initialiation code here

        my.travelWS.TravelWS service = new my.travelWS.TravelWS();
        my.travelWS.TravelService port = service.getTravelServicePort();
        my.travelW

        // <editor-fold defaultstate="collapsed" desc="Creator-managed Component Initializat
        // Initialize automatically managed components
        // *Note* - this logic should NOT be modified
```

```
init:
wsimport-init:
wsimport-client-check-TravelWS:
wsimport-client-TravelWS:
init:
wsimport-init:
wsimport-client-check-TravelWS:
wsimport-client-TravelWS:
wsimport-client-generate:
wsimport-client-compile:
```

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.

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:

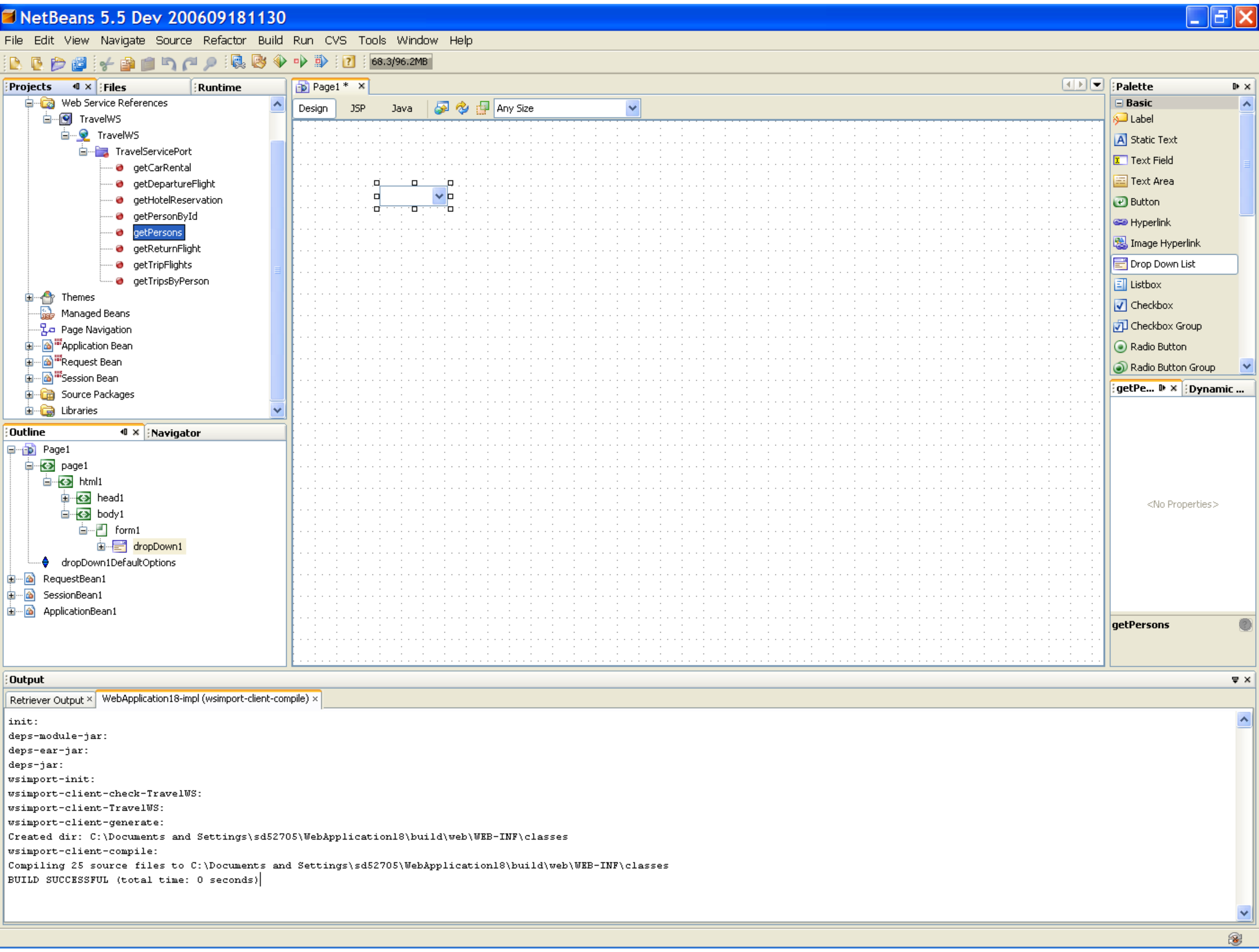
- There is no client instance added in the project yet when a method is dropped. In this case, a client instance will automatically be created and the data provider instance will be associated with it
- There is exact one client instance is in the project. Then the data provider instance will be associated 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 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.



```
init:
deps-module-jar:
deps-ear-jar:
deps-jar:
wsimport-init:
wsimport-client-check-TravelWS:
wsimport-client-TravelWS:
wsimport-client-generate:
Created dir: C:\Documents and Settings\sd52705\WebApplication18\build\web\WEB-INF\classes
wsimport-client-compile:
Compiling 25 source files to C:\Documents and Settings\sd52705\WebApplication18\build\web\WEB-INF\classes
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Drag the `getPersons` method from the Projects window and drop it on the Drop Down List.

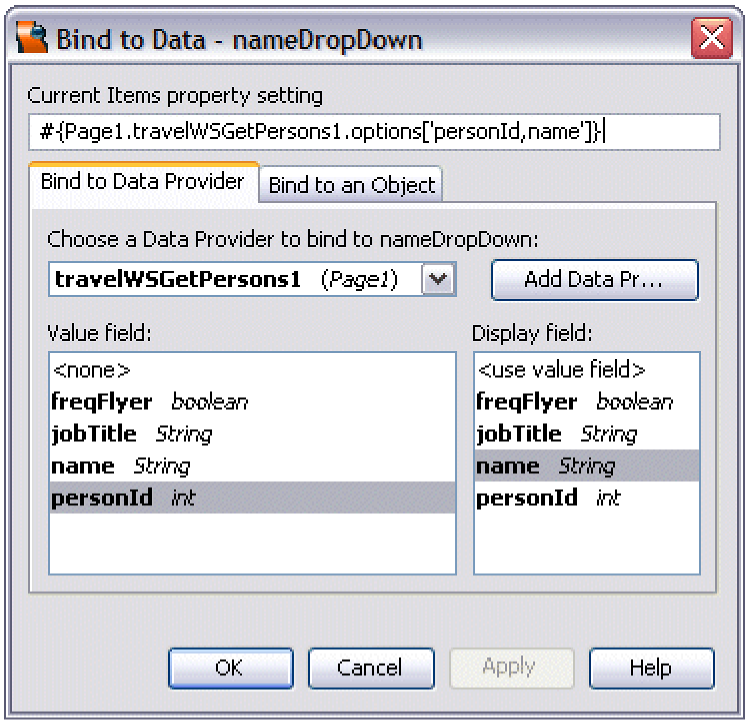
   The value in the Drop Down List changes from item 1 to abc. The `"abc"` text indicates that the display field is bound to a `String` object.

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


Bind to Data - nameDropDown
Current Items property setting
#{Page1.travelWSGetPersons1.options['personId,name']}
Bind to Data Provider
Bind to an Object
Choose a Data Provider to bind to nameDropDown:
travelWSGetPersons1 (Page1)

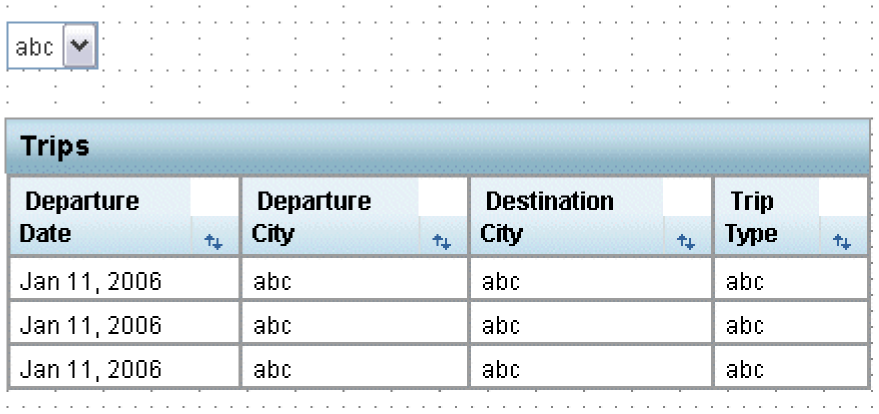
Value field:	Display field:
<none>	<use value field>
freqFlyer *boolean*	freqFlyer *boolean*
jobTitle *String*	jobTitle *String*
name *String*	name *String*
personId *int*	personId *int*

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.



Trips

Departure Date	Departure City	Destination City	Trip Type
Jan 11, 2006	abc	abc	abc
Jan 11, 2006	abc	abc	abc
Jan 11, 2006	abc	abc	abc

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.

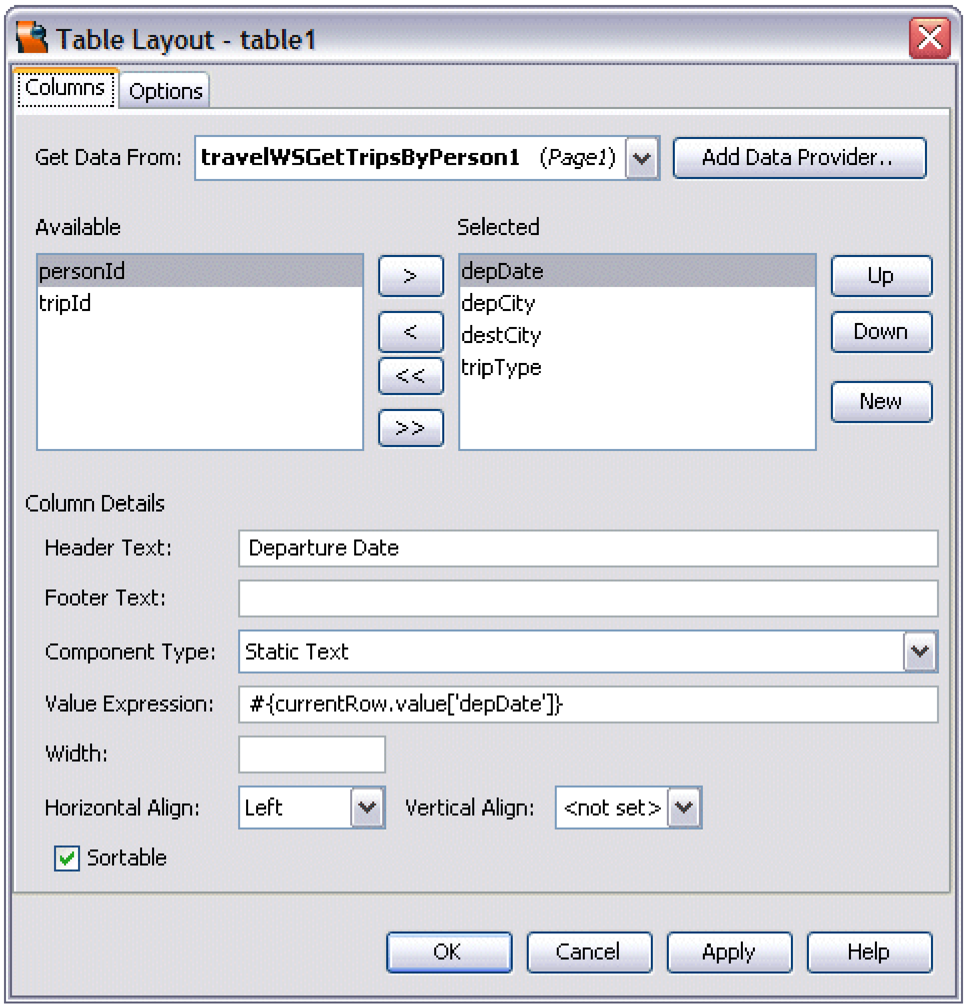
The value fields of the TravelWS data provider appear as the column names in the Table.

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.



Get Data From: travelWSGetTripsByPerson1 (Page1)

Available	Selected
personId	depDate
tripId	depCity
	destCity
	tripType

Column Details

Header Text: Departure Date
Footer Text:
Component Type: Static Text
Value Expression: #{currentRow.value['depDate']}
Width:
Horizontal Align: Left
Vertical Align: <not set>
[✓] Sortable

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**.

`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.