Build a rich Java client that uses a Web service

Contents

Build a rich Java client that uses a Web service

This tutorial teaches you how to use the Java visual editor to build a rich Java client that connects to a Web service. The client that you build in the tutorial is called My Company Directory.

My Company Directory is a Java application that is used to maintain a company's employee directory. The application connects to a sample Web service that provides methods for creating, retrieving, updating, and deleting employee records

The client is built visually in the Java visual editor using Swing components. The Java visual editor provides a set of helper classes (data sources, data objects, and binders) or connecting to and working with the Web service. The Web service is deployed locally on your own installation of IBM WebSphere Application Server v6.0, and the tools help you generate a Java proxy for your client based on a Web Services Description Language (WSDL) file.

See the finished product

Learning objectives

In this tutorial, you will learn the following lessons:

- How to use the Java visual editor to design and layout a user interface
- How to bind interface elements to data objects and a Web service

Time required

2 hours and 15 minutes

Related information

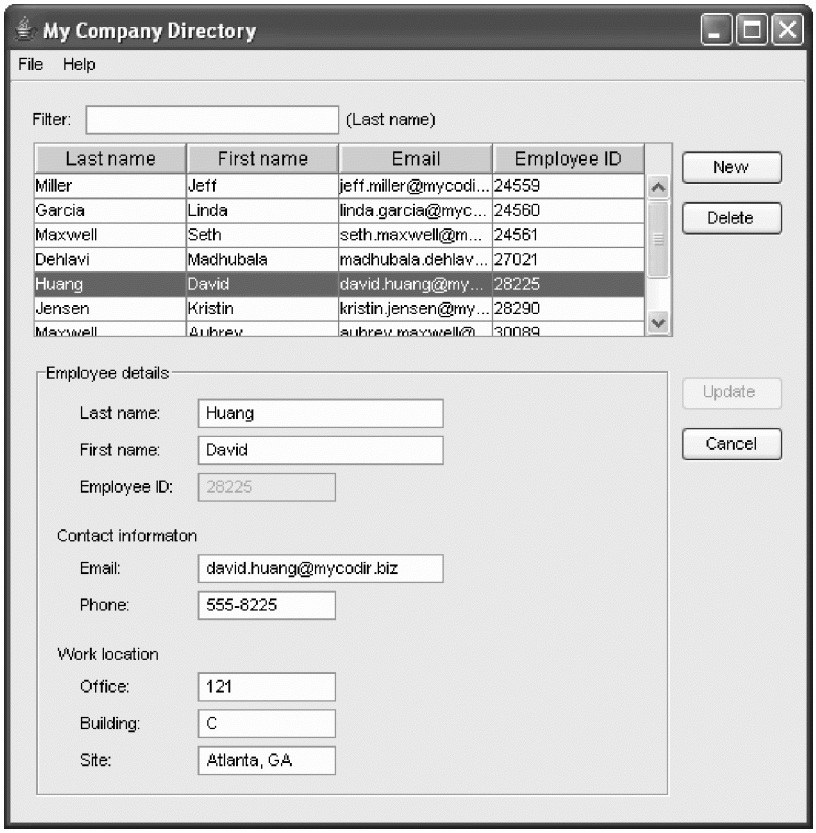
View the PDF version

Tutorial: Hello World Java

Introduction: Build a rich Java client that uses a Web service

The graphical user interface (GUI) for the client was mostly prebuilt for you visually using Swing components. In the first module, you will finish laying out the essential GUI components by using the Java visual editor. In the second module, you will bind the GUI components to the Web service data source, services, and objects returned by the data source. When building the application in the Java visual editor, you will use data sources, data objects, and binders, which are instances of helper classes that are generated by the Java visual editor and used by your application.

See a picture of the finished product:



Learning objectives

In this tutorial, you will learn how the following lessons:

- How to use the Java visual editor to design and layout a user interface
- How to bind interface elements to data objects and a Web service

Time required

To complete the entire tutorial, you will need approximately 2 hours and 30 minutes.

System requirements

- WebSphere Application Server v6.1. This server may already be installed with your product, or you can use your own standalone installation. The scenario in this tutorial asks you to deploy a sample Web service on the WebSphere Application Server that is running locally.

The sample Web service may run on other servers, but this tutorial has only been tested with WebSphere Application Server v6.0 and v6.1.

Prerequisites

You should be familiar with the following concepts:

- Basic Java development
- Basic Web service principles
- Basic workbench skills, such as working with projects and navigating perspectives and views

Module 1: Design the client GUI in the visual editor

This module teaches you how to use the Java Visual Editor to add a visual component to an application then visually lay it out and set arrangement constraints. The final lesson of this module shows you how to run the Java file to see how it will look as an actual application.

Remember: Before beginning this module, you should have the prerequisite knowledge outlined in the tutorial introduction.

Learning objectives

After completing the lessons in this module you will know understand the concepts and know how to do the following:

- Add and layout a JTable in a Java interface
- Run a visual class to test your work

Time required

This module will take approximately 15 minutes to complete.

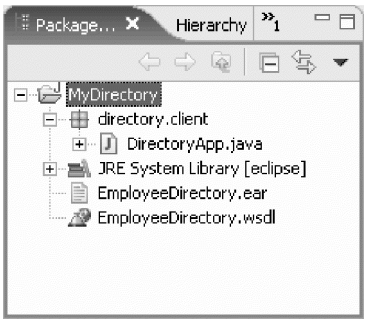
Lesson 1.1: Set up the Java project

In this lesson, you will set up the MyDirectory project by importing a project into your workspace. The project includes a single Java class, as well as other files that will be used later.

Because the main focus of this tutorial is binding visual components to a Web service, most of the Java GUI for the "My Company Directory" application has been designed for you already.

The MyDirectory project is the main Java project that you will work with in this tutorial. It contains your DirectoryApp.java file, which is the Java file that contains the main Java application that you are building. This tutorial includes several versions of the MyDirectory project to help you: one for the start of each module and a finished version of the completed project.

1. Import the MyDirectory project.
2. In the Package Explorer of the Java perspective, make sure that your MyDirectory project looks like the following image:



Lesson checkpoint

In this lesson you imported the example MyDirectory project, which acts as the starting point for this tutorial.

The MyDirectory project includes the following resources:

- DirectoryApp.java: A Java file that contains the application that you are developing in this tutorial. The DirectoryApp.java file is in a Java package named directory.client.
- EmployeeDirectory.ear: An enterprise application that contains the sample Web service. In Module 2, you will deploy this Web service on a local installation of WebSphere Application Server v6.0.

- EmployeeDirectory.wsdl: An XML file that uses the Web Services Description Language (WSDL) to describe the sample Web service that you will deploy. In Module 2, you will use this WSDL file to generate a Java proxy for your application to use.

Lesson 1.2: Add and lay out the employees table

In this lesson, you will use the Java visual editor to add a JScrollPane and a JTable to the application. In later exercises, you will program the JTable to get its data from a Web service that returns a list of all the employees in the company directory.

After you add the JTable, you will use the design view of the Java visual editor to customize the layout of the JTable to match the following specifications:

- Span the JTable across three cells horizontally and two cells vertically
- Add a left inset of 15 pixels
- Rename the JTable to employeesTable.

Show Me

Open the DirectoryApp.java file in the Java visual editor

To open the DirectoryApp.java file in the Java visual editor:

1. In the Package Explorer view of the Java perspective, expand the MyDirectory project and the directory.client package.
2. Right-click the DirectoryApp.java file, and select **Open With → Visual Editor**. The Java visual editor loads the Java class and displays the design on the graphical canvas area.

Tip:

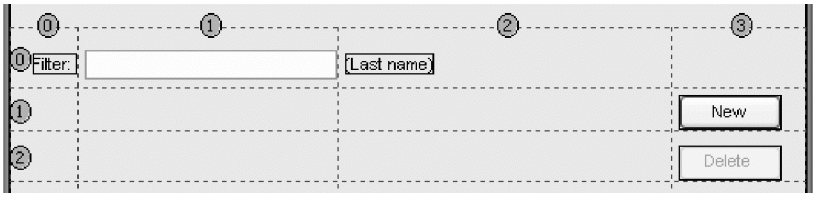
- To change the look and feel used by the Java visual editor, go to **Window → Preferences → Java → Visual Editor** and specify a Swing look and feel. The preference will take effect the next time you open the class. This tutorial uses the Windows look and feel.
- To make the Visual Editor the default editor for all Java files, you can click **Window → Preferences** and go to the **Workbench → File Associations** page to define your preference.

Add a JTable on a JScrollPane

The main window of DirectoryApp.java uses a JFrame with a JPanel for its main content pane. The JPanel in our application is named jContentPane. The jContentPane was set to use a type of layout manager called GridBagLayout. The GridBagLayout is a powerful layout scheme based on a grid of cells that can be occupied by visual components. The Java visual editor makes it easy to work with GridBagLayout by showing the grid borders. It also shows placement markers when you drop new components onto the grid, and it shows handles on components that you are resizing or moving on the GridBagLayout.

To add the employees table (a javax.swing.JTable) to the DirectoryApp.java user interface:

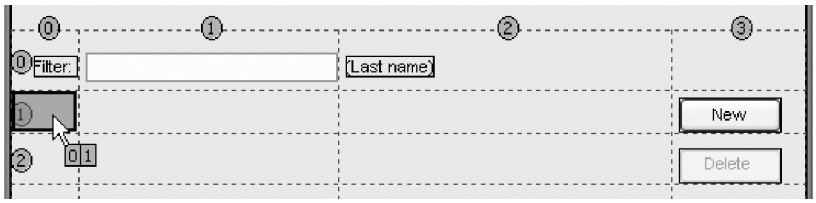
1. Right-click the jContentPane in the design view or Java Beans view, and select **Show Grid**. A red dotted line shows the grid border, and blue circles with numbers indicate the row and column numbers. For example, notice that the **New** button occupies the cell at row 1 (grid y) and column 3 (grid x).



2. In the Java visual editor palette, select the **JTable on JScrollPane** Swing component, which is categorized under the **Swing components** drawer of the palette.

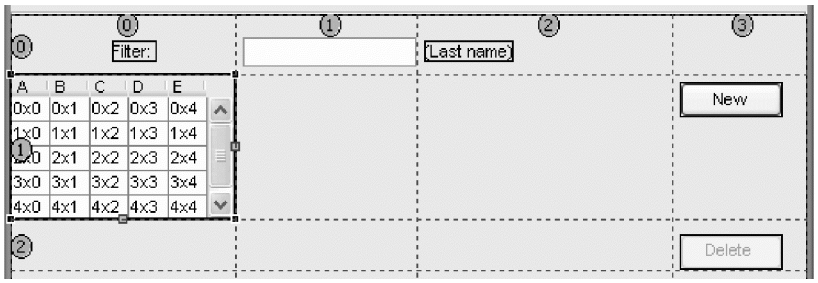
Tip: By default, the palette is collapsed on the right side of the design area. You can resize and move the palette.

3. Move your mouse pointer over the cell in the grid at column 0, row 1:



   - As you move your mouse pointer over the grid, the mouse pointer shows two numbered squares that tell you the x and y coordinates in the grid based on the location of your mouse pointer.
   - If you hover your mouse pointer directly on a grid border, new rows and columns can be created, and existing rows and columns will be renumbered. In this case, yellow squares on the mouse pointer, yellow bars between the grids, and yellow column and row labels indicate this behavior and point out the impact that the placement will have.

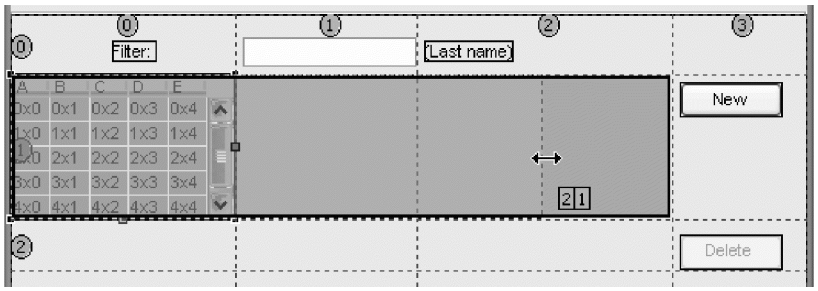
4. Left-click to drop the JScrollPane and JTable into the cell at column 0 and row 1:



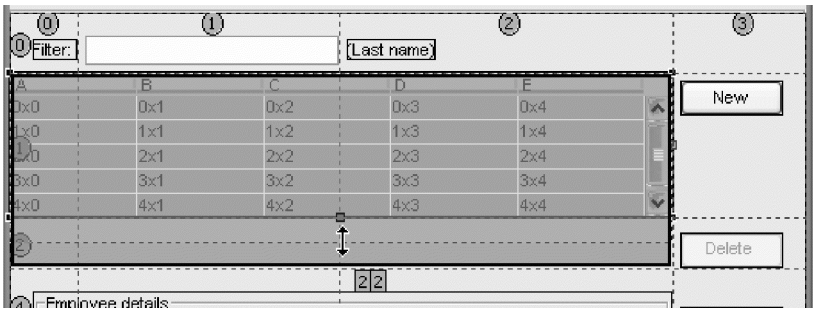
Span the JScrollPane and JTable across multiple columns and rows of the grid

Now you need to make your JScrollPane (and its child JTable) span three columns and two rows for better spacing and resizing behavior. To make the table span the columns and rows:

1. Select the JScrollPane in the design area or Java Beans view (it should still be selected because you just added it). Notice the small green squares on the right and bottom of the JScrollPane. You will use these resize handles to drag the JScrollPane to span multiple columns and rows.
2. Click and hold down your left mouse button on the green handle on the right side of the JScrollPane.
3. Drag your mouse pointer to the right until the placement indicates column 2, row 1. A dark gray shadow will also indicate the cells that the component will occupy when you release the mouse button.



4. Release the mouse button. The JScrollPane now spans the three columns.
5. Repeat the similar process to drag the bottom handle of the JScrollPane until the JScrollPane spans into row 2:



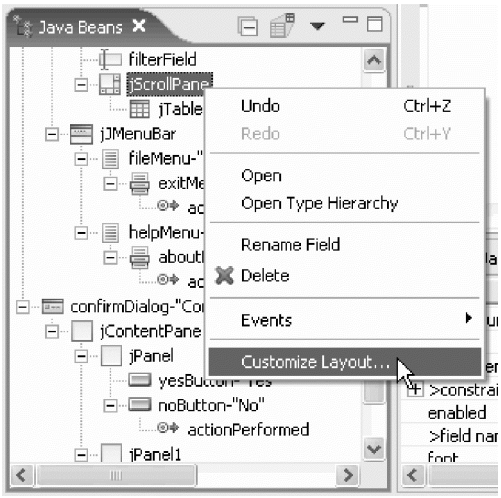
Customize the spacing of the JScrollPane within the GridBag

Another feature of the GridBagLayout manager is that you can specify various constraints to further customize the layout. For example, you can specify the following constraints:

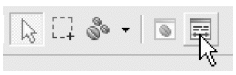
- **anchor:** A component can be given an anchor orientation within its cell, which will affect how the component moves as the application is resized by a user. For example, a component could be anchored at top-left, middle-left, center, or bottom-right.
- **fill:** A component can be told to occupy all available space within its cell or cells either horizontally, vertically, or both.
- **insets:** A component can be given its own padding on the top, bottom, left, and right side to provide spacing between the component and the edge of the grid.

To customize the anchor, fill, and insets for the JScrollPane:

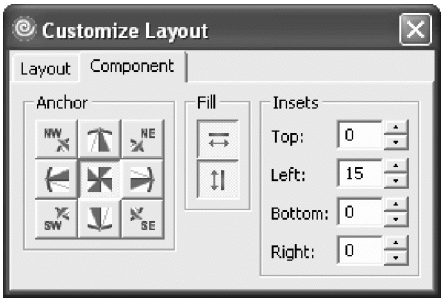
1. Right-click the JScrollPane in the design view or Java Beans view, and select **Customize Layout**.



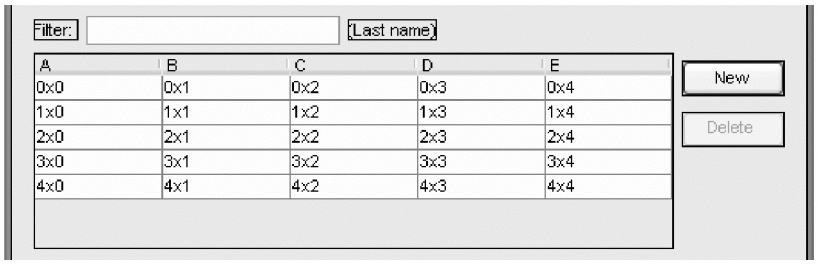
Tip: The Customize Layout dialog box can remain open as you select and change the layout for different components. You can open the Customize Layout dialog box at any time by clicking the Customize Layout button in the menu bar:



2. On the Component tab of the Customize Layout dialog box, make sure that the Anchor center button is pressed.
3. Make sure that both the **Fill horizontal** and **Fill vertical** buttons are pressed.
4. Add a left inset of 15 (pixels) to make the spacing on the left side of the JScrollPane similar to the other visual components on the application.



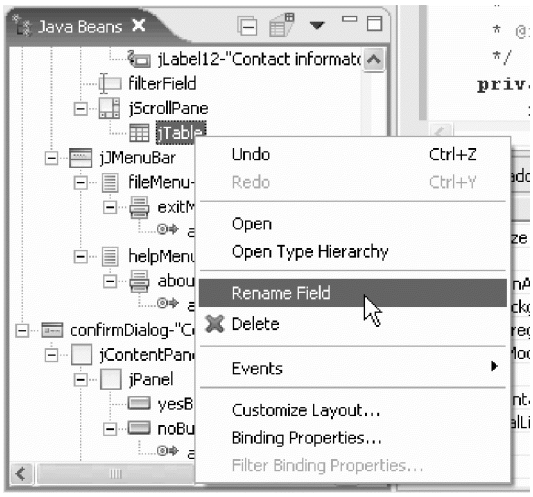
The table now aligns with the **Filter** label, for example.



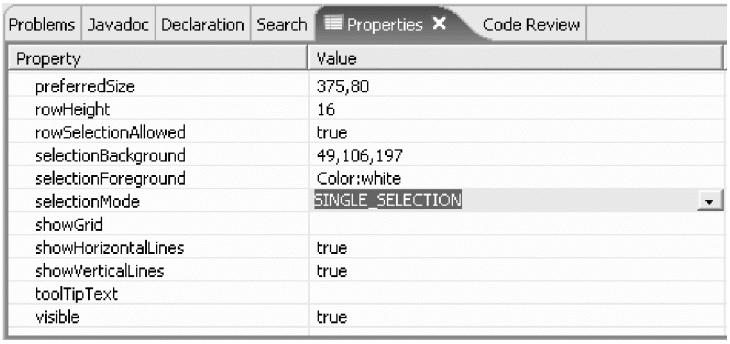
Rename the new JTable to a useful value and set it to select a single row

Because you will later work with the table, it will be useful for you to rename the JTable instance and its getter method. To rename the table:

1. In the Java Beans view, right-click the jTable component and select **Rename field** from the pop-up menu.



2. Type `employeesTable` and click **OK**. The JTable is now named employeesTable, and the method for instantiating it is getEmployeesTable.
3. Set the table to allow only a single row to be selected:
   a. Select the employeesTable in the design view.
   b. In the Properties view, select the **selectionMode** property and set it to SINGLE_SELECTION.



c. Save the DirectoryApp.java file.

Lesson checkpoint

In this lesson you learned how to use the visual editor to add a table to an existing user interface. Then you learned how to customize its layout, positioning, and spacing.

Lesson 1.3: Run the visual class

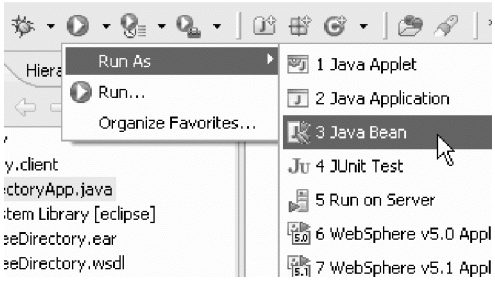
Now you are ready to run the Java application to preview its appearance. The workbench and visual editor make it very easy to quickly run your application, and you can repeat these steps at any time in your development to test the actual run-time appearance and behavior of the class.

Show Me

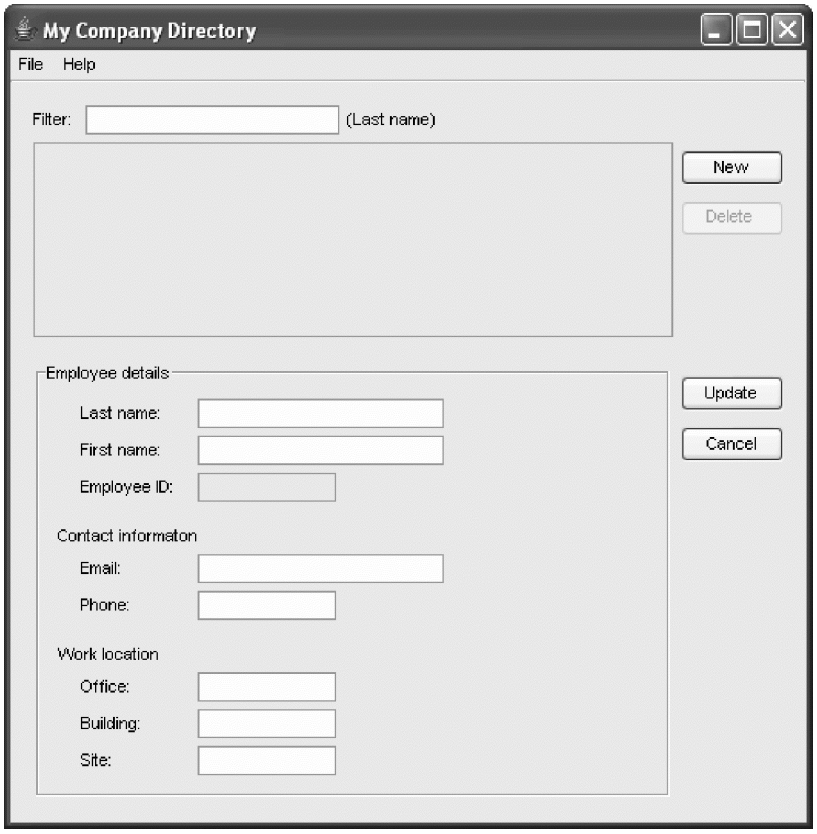
The Java visual editor provides a Java Bean launcher that is capable of running classes with no main() method. When it runs the visual class, it launches the application in a separate virtual machine (VM). If you run a visual class as a Java application, the launcher attempts to execute the main() method in the class. For this tutorial, your application includes a main() method that invokes and shows the DirectoryApp JFrame, so you can run it as an application or as a Java bean.

To run the DirectoryApp.java file as a Java bean:

1. Make sure that your DirectoryApp.java file is open in the Java visual editor.
2. From the menu bar, click **Run** → **Run As** → **Java Bean**.



Tip: The application opens on your desktop using the Swing look and feel that you have defined in your Visual Editor preferences (**Window** → **Preferences** → **Java** → **Visual Editor**). Alternatively, you can click **Run** → **Run**, and define the look and feel for the particular launch configuration for launching this Java bean. If you run this application as an application rather than a bean, it will also use the Windows look and feel because it is defined in the main() method. The screen shots used in this tutorial show the Windows look and feel.



Lesson checkpoint

Because you have only designed the interface but have not programmed any data connection or event functionality, you cannot do anything with your application. However, you can see the basic layout and appearance as it will look to a user. You can try clicking some of the buttons, but you will notice that they do nothing. The File menu and Help menus, however, are already implemented for you. You can try them to see what they do, and you can inspect the Java code to see how they are implemented with actionPerformed events.

Lessons learned

This module introduced you to designing the interface for a rich client using the Java visual editor. Beyond designing the visual appearance of a client, however, there is much more that you need to do to actually make the client useful. You will typically need to include event behavior or other logic and, in this case, the binding of the visual elements to a data source of some sort.

In this module, you learned how to perform the following tasks:

- Import a Java project using Project Interchange import
- Add a JTable on a JScrollPane to your visual class
- Use the GridBagLayout manager to visually lay out the table on the rich client
- Run the application to see the actual appearance of the rich Java client

In the next module, Module 2: Bind visual components to the Web service, you will take the simple My Company Directory interface and turn it into a powerful rich client that accesses Web service methods for creating, retrieving, updating, and deleting employee records from a company directory.

Module 2: Bind visual components to the Web service

This module teaches you how to bind the visual elements of My Company Directory (the buttons, the employee table, fields, and other actions) to a Web service. The Web service provides the real functionality to create, retrieve, update, and delete employees from the sample directory.

Learning objectives

After completing the lessons in this module you will know understand the concepts and know how to do the following:

- Bind a table to a data Web service data source
- Bind fields to objects
- Program buttons with actions

This module will take approximately **2 hours** to complete.

Lesson 2.1: Install and deploy the Web service

In this exercise, you will install a sample enterprise application (EAR) file onto WebSphere Application Server v6.1 and deploy the EmployeeDirectory Web service. Your application will use this Web service to create, read, update, and delete employee records.

Before you begin, you must complete *one of the following options* to make sure that your MyDirectory project is at the proper starting point:

- Complete "Module 1: Design the client GUI in the visual editor" on page 3.

  or

- Import the MyDirectory project at Module 2 starting point

  Tip: Unless you specify a different project name during import, this will overwrite your MyDirectory project contents.

Your MyDirectory Java project includes an EmployeeDirectory.ear file. You will use the WebSphere Administrative Console to install the EmployeeDirectory enterprise application that is contained in the EAR file. When you install the application, you also deploy the Web service included in the application. The finished My Company Directory application uses this deployed Web service.

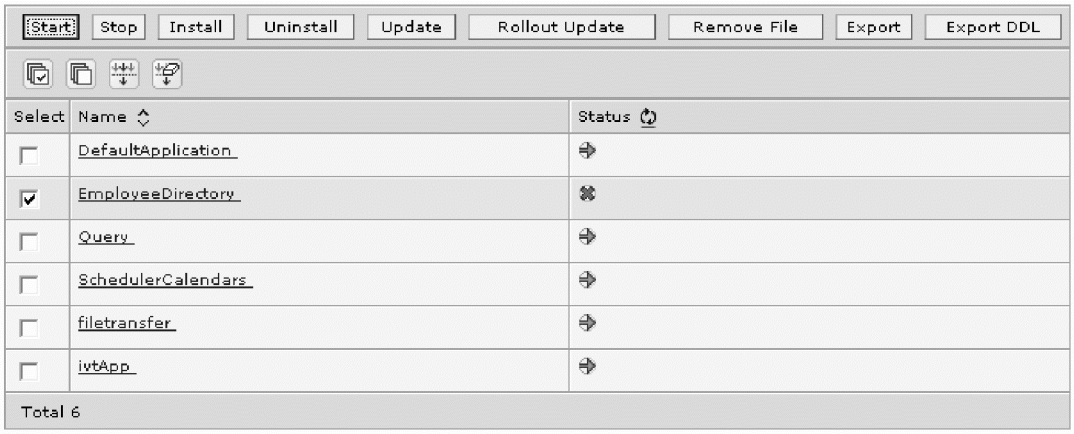
To install the sample EmployeeDirectory application and deploy the Web service on your WebSphere Application Server v6.1 environment:

1. Start an instance of your application server from the workbench. There are several different ways that you can launch your server, but these steps describe how to do it from the workbench:
   a. Open the Servers view. To add the Servers view to the Java perspective, click **Window → Show view → Other and select Server → Servers**.
   b. Tthe Servers view lists the servers that are installed and set up.
   c. Right-click your server and select **Start**. When the Servers view shows the status of the server as **Started** or the console says `Server server1 open for e-business`, the server is successfully started. You can now run the Administrative Console.

   Note: If there is no server instance in the Servers view, create a new server:
   a. Right-click in the Servers view and select **New → Server**.
   b. Use the New Server wizard to add WebSphere Application Server v6.1.
2. Run the WebSphere Administrative Console. Again, there are other ways to run the Administrative Console, but these instructions describe how to do it from the workbench:

a. In the Servers view, right-click the server that you just started, and select **Run administrative console**. The WebSphere Administrative Console opens in a browser window.
b. Enter a user ID and click **Log in**. The Welcome page of the Administrative Console opens. The user ID that you enter is only used to track user-specific changes to the configuration data of the server.

3. Use the Administrative Console to install the EmployeeDirectory.ear enterprise application that is found in your MyDirectory project. The Administrative Console uses a wizard approach to help you install applications, where you click **Next** to move from page to page until all options are set. To install the sample enterprise application that contains the Web service for this tutorial:
   a. On the left side of the Adminstrative Console, expand the **Applications menu** option, and click **Install New Application**.
   b. Select **Local file system** and in the **Specify path** field enter the full path to the EmployeeDirectory.ear file that is in your MyDirectory project. Tip: To get the full path, right-click on the EmployeeDirectory.ear file in the Package Explorer and select **Properties**. The Properties page lists the location of the file, which you can copy and paste into the **Specify path** field.
   c. Click **Next** until you reach the **Select installation options** page.
   d. Select **Deploy Web services**.
   e. Click **Next** until you reach the **Summary** page, then click **Finish**.
   f. Click the **Save to Master Configuration** link when you are prompted to apply the changes that you have made to your local configuration. Review the changes and click the **Save** button.
4. Use the Administrative Console to start the EmployeeDirectory application:
   a. Click **Applications → Enterprise Applications**. The EmployeeDirectory application is listed as an installed application on the server, but its status is Stopped.

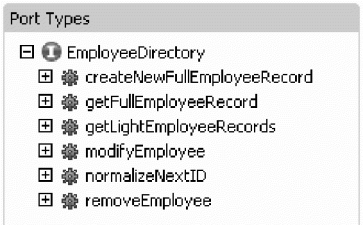


   b. Select the check box next to EmployeeDirectory and click **Start**. A message indicates that the EmployeeDirectory application started successfully, and the Status icon changes to the green arrow.

The EmployeeDirectory application is now running on localhost at port 9080, and the Web service can now be accessed. After you complete this tutorial, you can go back to the Administrative Console, stop the EmployeeDirectory application, then uninstall it.

If you open the EmployeeDirectory.wsdl file found in your MyDirectory project (it should open in the graphical WSDL Editor by default), you can examine the Web service that you just deployed. If the WSDL file does not open in the WSDL Editor, the Web Service Developer capability might not be turned on in the workbench. You can specify workbench capabilities in the Preferences (**Window → Preferences → Workbench → Capabilities**).

The following image from the WSDL editor shows the operations available in the EmployeeDirectory service:



You can use the WSDL editor to examine each operation and its corresponding request messages and return messages. This can help you understand the Web service and how it is used in the remaining exercises.

Lesson 2.2: Bind the employees table to the Web service data source

The My Company Directory application displays a list of all current employee records in the directory. The records are displayed in a JTable (employeesTable) with sortable columns, including last name, first name, email, and employee ID. In order to get the records for the table, you need to bind the employeesTable to a data object that is returned by the sample Web service data source.

Show Me

Overview of data objects, data sources, and binders

In order to get a local data object for the employeesTable to work with, you will use the visual editor to add a data source to your application. The data source connects to the sample Web service proxy and discovers the service methods available to your application. You will then choose the getLightEmployeeRecord service method that is made available from the data source. Finally, you will bind the employeesTable in your application to the fields that are returned in the row data object (lightEmployeeRecordRows).

You can create all of these data sources and data objects quickly and easily by using the Java visual editor's built-in binder classes. The visual editor provides a set of generic interfaces and classes that are generated into your project as you bind visual components to data factories. The binder classes are generated by default into a package named jve.generated. The visual editor provides the binder classes as a generic implementation that you can further customize and enhance to meet your application's needs. This tutorial demonstrates the power and flexibility of even a basic and simple use of the default binder classes.

Important: Before you begin this exercise, it is highly recommended that you read the following help topics. These topics can help you learn more about the functionality and logic behind the data objects, data sources, and binders provided by the Java visual editor:

- Data binders overview
- Binder API reference

For this tutorial, you will use a Web service data source, several types of data objects, and several types of binders in your application. When you add instances of these objects to your application, the visual editor adds the necessary classes into the jve.generated package in your project, where you could extend, replace, or rewrite the data binding logic. The Java visual editor provides visual support for the binding objects by showing on the free-form area of the design view the data objects, data sources, and binders that your application is using. The visual editor draws lines between visual components and the data objects and data sources to show the current bindings for any selected object.

The following diagram is a simple overview of how visual components, binders, data objects, and data sources interact. The application that you build in this tutorial illustrates a slightly more complex and creative use of the binders. This diagram does not represent exactly the binders, data objects, and data sources in the sample application that you are building.

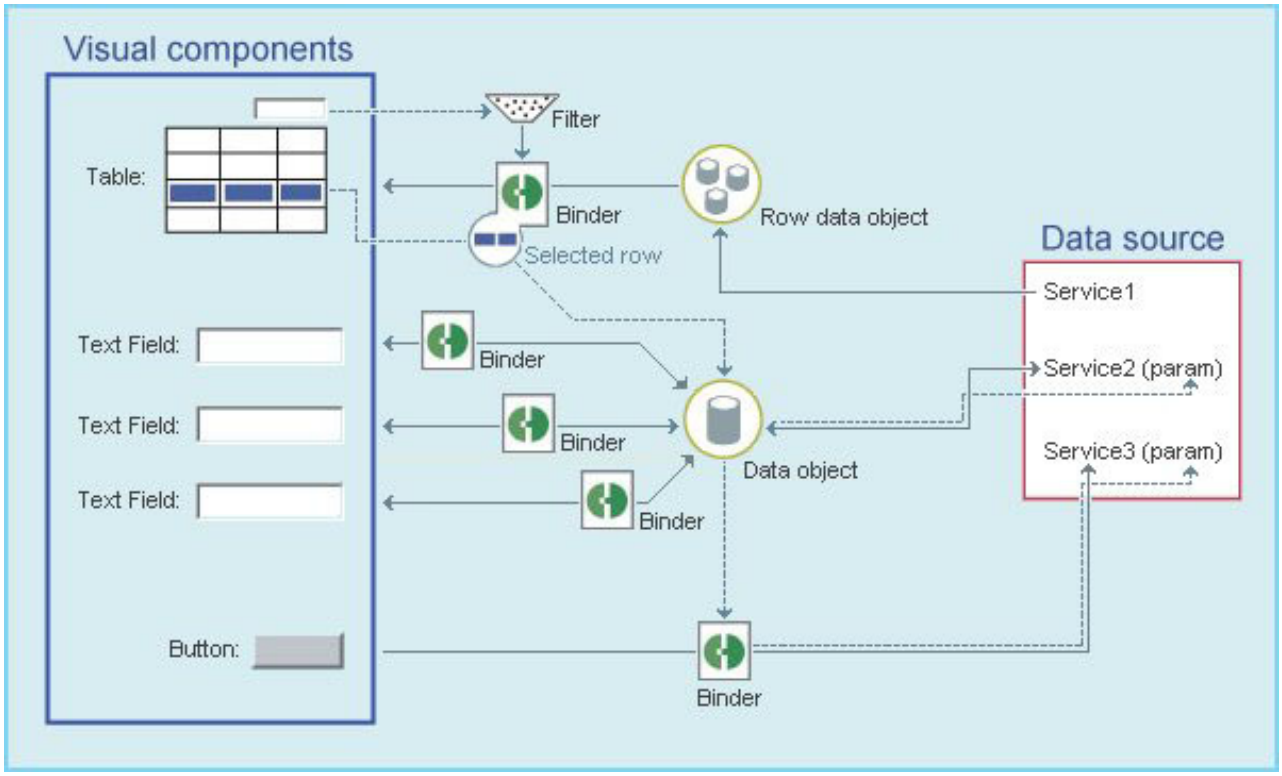


Figure 1. This diagram illustrates a sample relationship between visual components, binders, data objects, and data sources

In Figure 1, each visual component has its own binder that associates it with a data object or, in the case of the button, a data source. The binders for the text fields bind the field to a particular property of the data object. Both the row data object and the data object in this diagram get their data from direct calls to a service on the data source. The data object for the text fields uses a key value from the selected row in the table as its argument for calling Service2, which returns a full record that presumably includes more information about the selected row in the table. This full record, in turn, is used as the argument for the button's action binder when it calls Service3, which could be a method that updates the values entered in the fields. For more detailed explanations of the data objects, data binders, and data sources, follow the links provided earlier.

Generate a Web service Java proxy in your project using the provided WSDL file

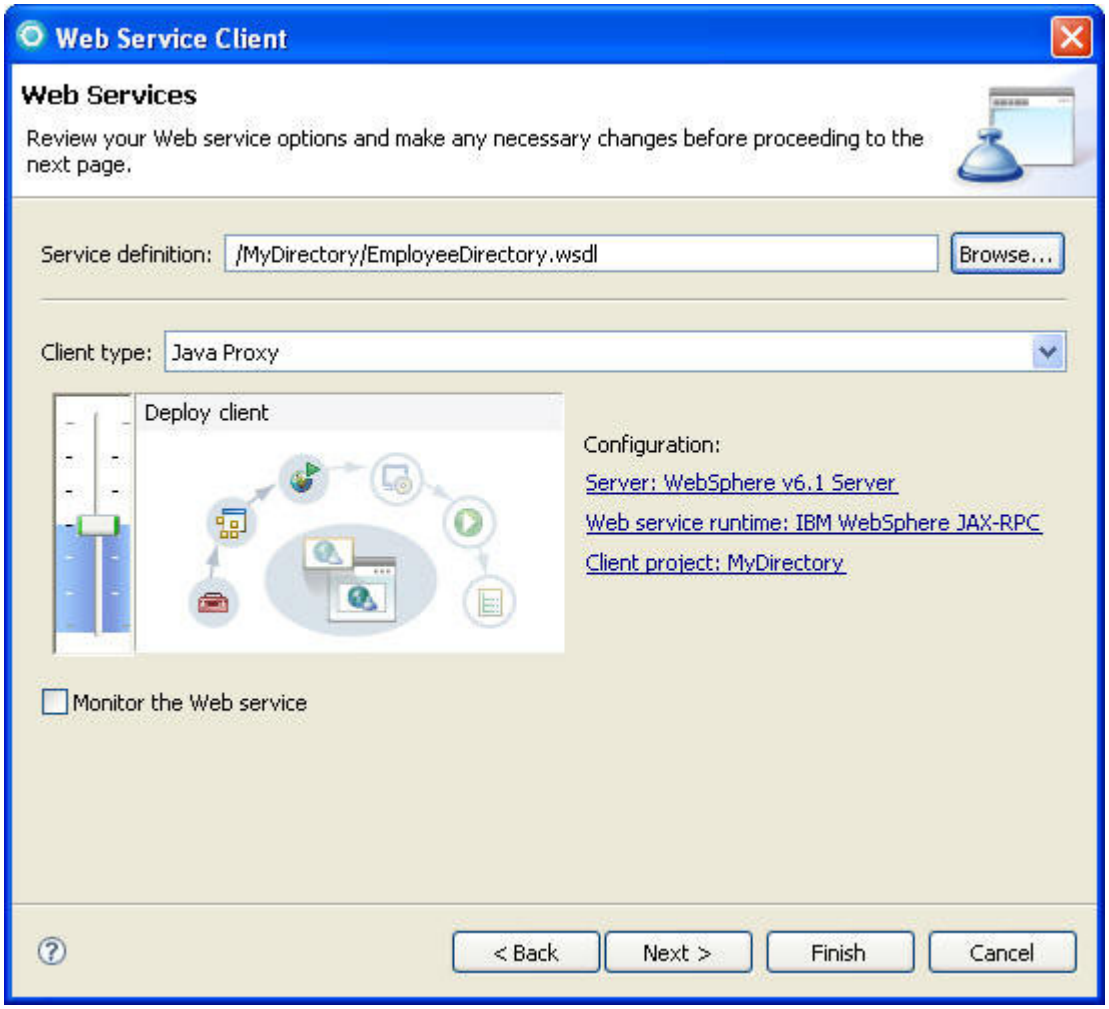
In order to work with the Web service running on a server, your Java application requires a Java proxy, or client, to interact with it. Using a WSDL file, you can generate a Java proxy into your Java project using the Web Service Client wizard. Your MyDirectory project includes the EmployeeDirectory.wsdl file that you will use to generate this proxy. After you generate the Java proxy, you can create a data source that represents the Web service and begin binding visual components.

Important: The WSDL file that is used in this exercise assumes that you deployed the Web service on a local installation of WebSphere Application Server and used the default port for localhost (http://localhost:9080). If you deployed the EAR file differently, you must edit the WSDL file accordingly before you proceed.

To generate the Web service Java proxy in your project:

1. On the main menu, click **File** → **New** → **Other** and select the **Web Services** → **Web Service Client** wizard. If the Web Services category is not showing, select **Show all wizards**.
2. Use the wizard to define the Web service client:

a. For the **Service definition**, enter the WSDL file that is provided in your MyDirectory project: `/MyDirectory/EmployeeDirectory.wsdl`
b. In the **Client type** field, select **Java proxy**.
c. Set the slider bar to **Deploy client**.
d. Make sure the server is and Web service runtime are set properly for the server you are running. This tutorial has been tested against WebSphere v6.0 and WebSphere v6.1 with the IBM WebSphere JAX-RPC runtime.
e. Make sure the Java proxy client is output to the MyDirectory project.



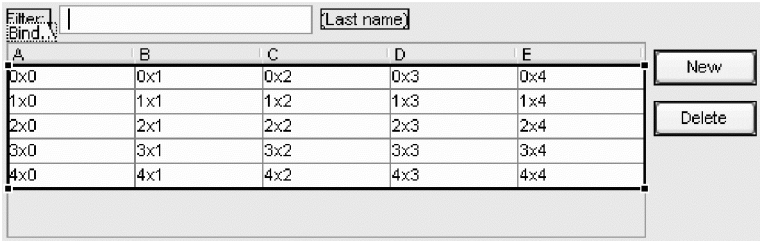
3. Click **Finish**. The Web Service Client wizard generates the Java proxy in a new package (directory.service) in your project.

Bind the employeesTable to a row data object returned by the Web service

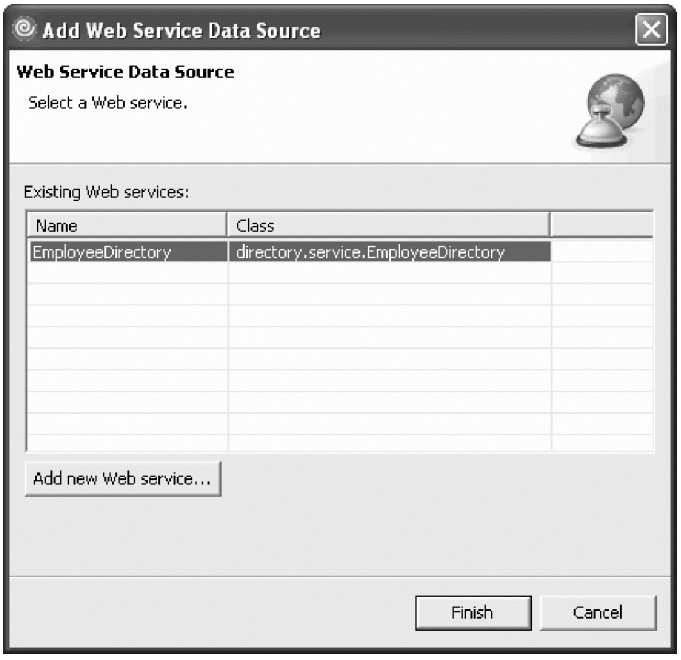
Because the employeesTable is the first visual component that you are binding in this application, you need to create a data source that points to the sample Web service proxy that you just added to your project. When you bind other visual components in later exercises, you will reuse this data source. In this step, you add the Web service data source and the lightEmployeeRecordRows data object.

To bind the employees table:

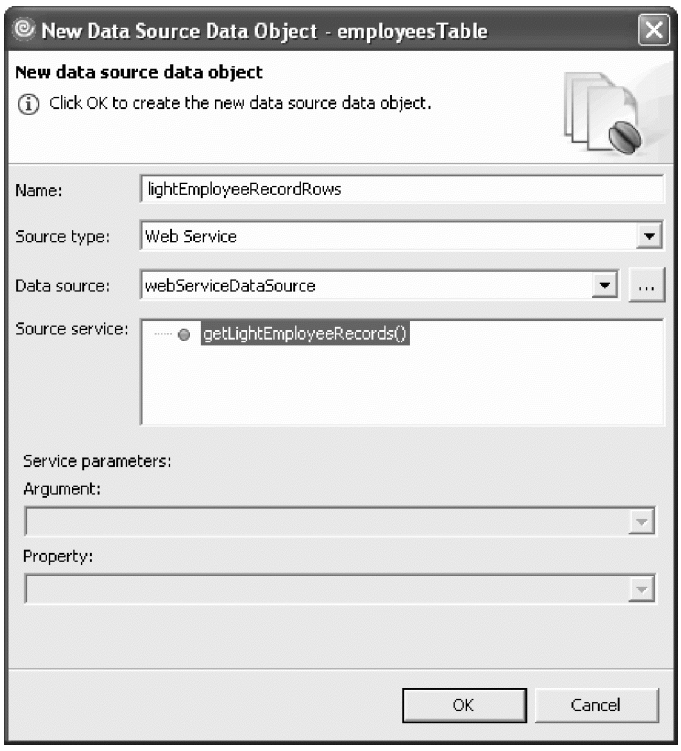
1. In the Java Beans view or design view, select the employeesTable. (Make sure that you do not select its JScrollPane parent.) A small tab labeled **Bind** shows on the top of the employeesTable in the design area.



2. Click the **Bind** tab on the employeesTable. Alternatively, you can right-click the employeesTable and select **Binding Properties**.
3. Because there are no data objects in your application, you need to add a new one. Click **New Data Source Data Object**.
4. In the **Source type** field, select **Web service**.
5. Because you have not yet added the Web service data source to your application, you need to add it now. Next to the **Data source** field, click the **...** button to open the Add Web Service Data Source dialog box, which looks for available Web service clients, or proxies, in your project.
6. Select the EmployeeDirectory Web service and click Finish. A new data source is added to the DirectoryApp.java file.

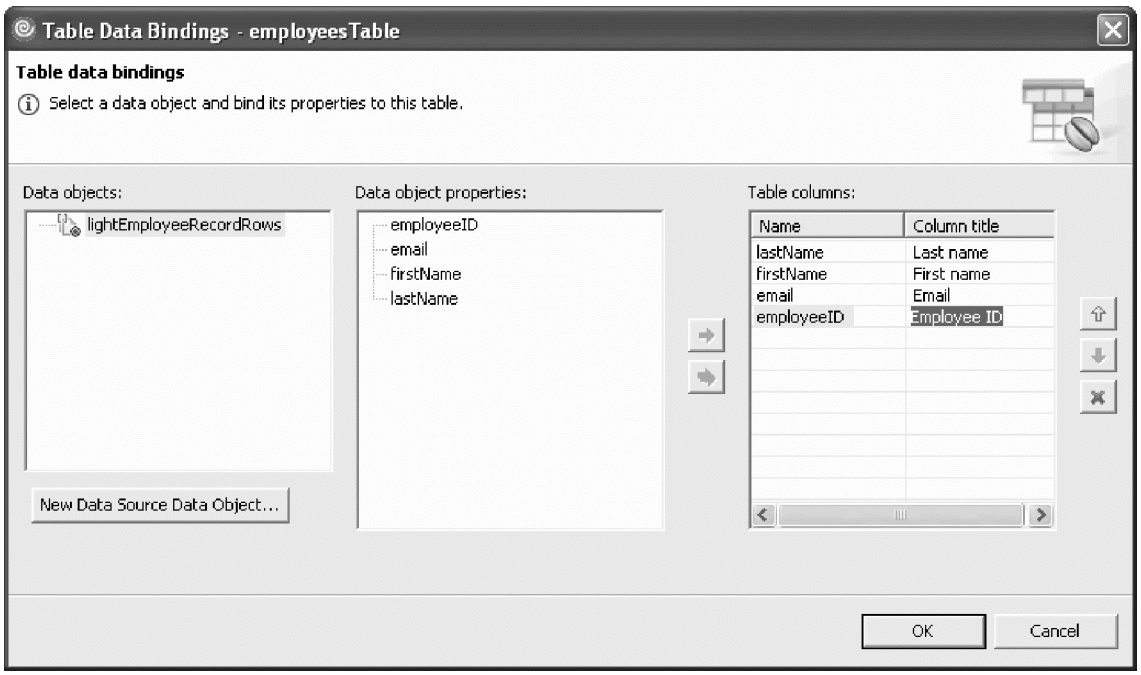


7. On the New Data Source Data Object dialog box, select getLightEmployeeRecords() in the Source service field, and accept the default name for the new data object: lightEmployeeRecordRows. No parameters are needed for this service method. Click OK. The new data object is created and displayed on the free-form area of the design view.



Tip: Because you are binding a table, the New Data Source Data Object dialog box only displays services that return row data objects. In this case, the getLightEmployeeRecords() method is the only service available that returns an array of objects.

8. On the Table Data Bindings dialog box, select the lightEmployeeRecordRows data object.
9. Now, you need to select the properties of the lightEmployeeRecordRows data object that you want to display on employeesTable:



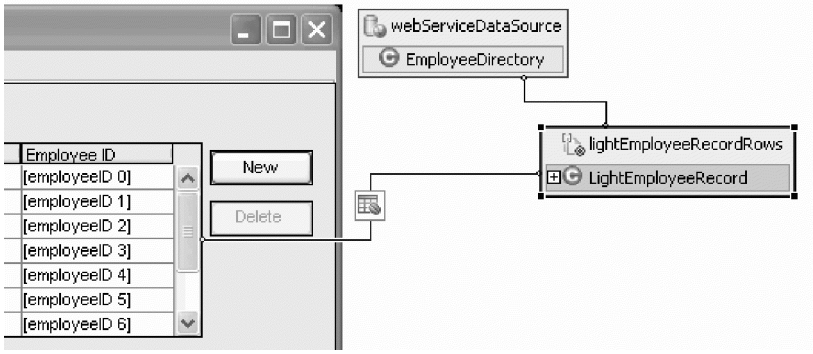
a. Click the double arrow ⇨ button to add all of the object properties to the **Table columns** list.
b. Use the up and down arrows to arrange the columns in the following order, top to bottom: lastName, firstName, email, employeeID

c. Rename the column titles: `Last name`, `First name`, `Email`, `Employee ID`

   Tip: After you finish binding the table, you can always go back to the binding properties and rename and reorder the columns at any time.

d. Click **OK**.

The employeesTable is now bound to the lightEmployeeRecordRows data object using a JRowTableBinder. If you click the lightEmployeeRecordRows data object on the free-form area, the visual editor draws a line from the data object to the table. On the line, the JRowTableBinder is represented by the table binder icon. Another line indicates that the data object uses the webServiceDataSource as its data source.



Lesson checkpoint

Notice the changes to your project and application. During this lesson you added the Web service data source, a row data object, and a binder that binds the employeesTable to the row data object.

Examine the new package (jve.generated) that was created in your project to hold all of the binder classes generated by the Java visual editor. Also notice the new package (directory.service) that holds the Java proxy for the Web service.Describe or summarize what was learned in this lesson.

Now, when you run the My Company Directory application, the employees table is populated by the Web service with the existing employee records.

Lesson 2.3: Bind the detail fields to the table selection

In the previous exercise, you bound the employeesTable to the lightEmployeeRecordRows data object returned by the getLightEmployeeRecords() service in the Web service. Now you need to populate the details fields based on the employee that is selected in the table.

To get the extra details for each selected employee, another data object is used. The selectedEmployeeRecord data object that you will add is returned by the getFullEmployeeRecord() service. This service takes the ID of the selected employee in the table as a parameter, and it fetches additional details about the employee, including phone number and work location.

The JRowTableBinder that was used when you bound the table to the row data object simplifies this step. The JRowTableBinder exposes the selected element in the table as a separate data object that can be used as the parameter for the getFullEmployeeRecord(java.lang.Integer) method. You can then easily bind each of the text fields to its corresponding property in the selectedEmployeeRecord data object.

Learn more about this Web service: The Web service includes two services for getting all the details of each employee. The table lists all employees, and only a subset of data is displayed in the table. Then, when a single employee is selected, you can retrieve the rest of the employee information for that selected employee only. If the Web service sent all data for each employee when the table requested data, the Web traffic could be heavy and cause slower performance of the application.

For example, if the employee record included a photo or an attachment, you would not want to retrieve all photos when you are simply getting the full list of employees. So, the getLightEmployeeRecord service is used to populate the table, and the getFullEmployeeRecord gets the full record for the employee who is selected in the table.

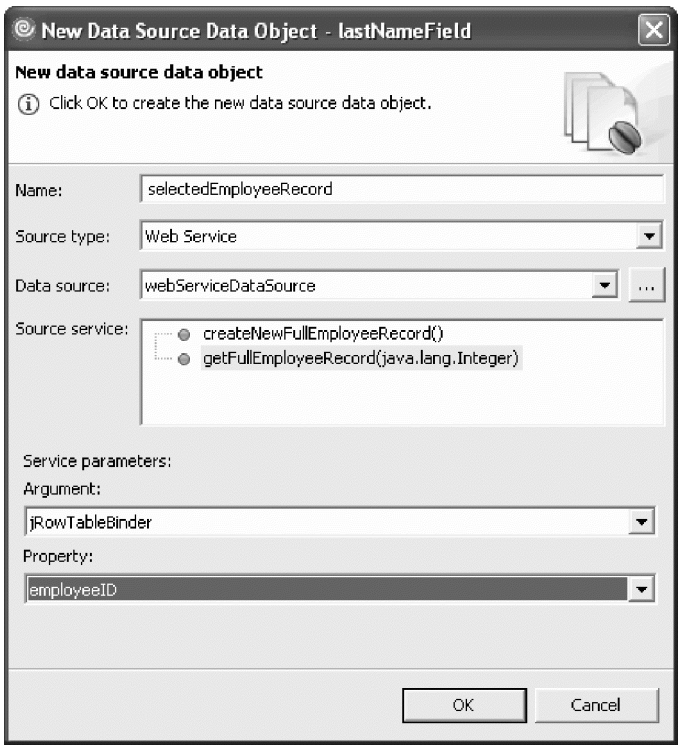
Bind the Last name field

In this step you will bind the **Last name** field to the lastName property in the selectedEmployeeRecord data object:

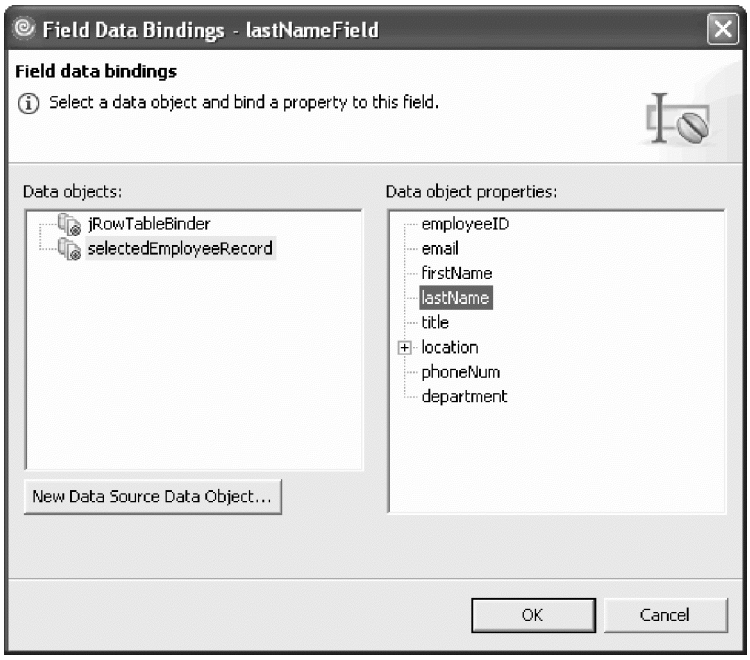
1. In the Java Beans view or the design view, select the JTextField for the last name (lastNameField). The design area shows a **Bind** tab on the text field.

2. Click the **Bind** tab to open the Field Data Bindings dialog box.
3. Click **New Data Source Data Object**. Although the existing jRowTableBinder data object does return the correct last name, it does not include the full employee record. You need to create a new data object that represents the full employee record.
4. In the **Source type** field, make sure that **Web Service** is selected, and for **Data source** make sure that **webServiceDataSource** is selected.
5. In the **Source service** list, select getFullEmployeeRecord(java.lang.Integer). The New Data Source Data Object dialog box lists the services that return data objects that are compatible with a text field.
6. In the **Name** field, enter `selectedEmployeeRecord`.
7. In the **Argument** field, select jRowTableBinder, and in the **Property** field, select employeeID. The employee ID of the selected row is now set to be the argument for the getFullEmployeeRecord() service method.

   Note: The getFullEmployeeRecord(java.lang.Integer) requires an integer as an argument. You want to use the employee ID of the current selection in the employees table to retrieve a full record. When you bound the table, the visual editor automatically generated jRowTableBinder, which listens for the current selection on the employees table. For the integer parameter, you will use the employeeID of the selected row in jRowTableBinder.

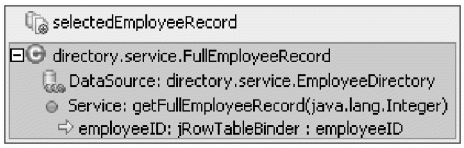


8. Click **OK**.
9. On the Field Data Bindings dialog box, make sure that selectedEmployeeRecord is selected in the **Data objects** list. Notice that there are more available properties for the selectedEmployeeRecord data object than for the jRowTableBinder data object.
10. In the **Data object properties** list, select the lastName property.



11. Click **OK**. The last name field in your application is now bound to the lastName property of the selectedEmployeeRecord data object, which is returned by getFullEmployeeRecord().

    A new data object named selectedEmployeeRecord is created and added to your application. A visual representation of the data object is added to the free-form area of the design view, as shown in the following image:



Now, when you select the lastName field on the design area, a line indicates that it is bound to the selectedEmployeeRecord. In the middle of the line the text binder icon represents the SwingTextComponentBinder that is used for this binding. If you select the line or the icon representing the binder on the design area, you can examine the binder's properties in the Properties view.

Bind the remaining details fields

To bind each of the remaining details fields for an employee, you follow a similar process as the last name field, but you do not need to add the data object. Because you already added the selectedEmployeeRecord data object, you can simply bind each field to its corresponding property in the selectedEmployeeRecord data object.

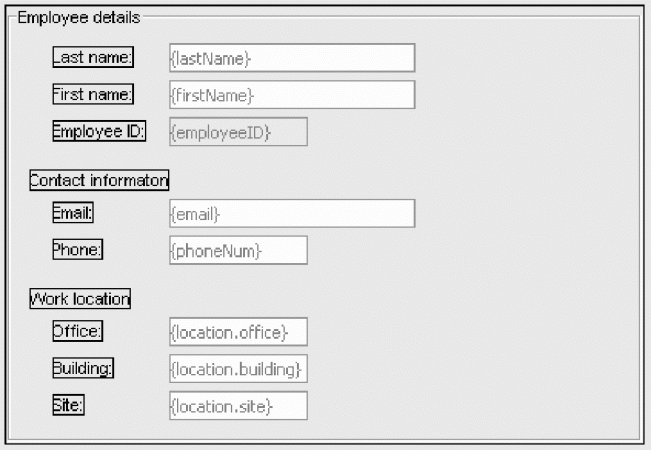
To bind the fields, complete the following steps for each of the fields in the Employee details section of the application:

1. Select the field in the design view, and click the **Bind** tab.
2. In the Field Data Bindings dialog box, select selectedEmployeeRecord from the **Data objects** list.
3. In the **Data object properties** list, select the appropriate property for the field that you are binding. The following chart shows the property that each text field needs to be bound to:

Field	**Property in selectedEmployeeRecord data object**
lastNameField	lastName
firstNameField	firstName
idField	employeeID
emailField	email
phoneField	phoneNum
officeField	location.office
buildingField	location.building
siteField	location.site

4. Click **OK**.

When you finish binding the text fields, the design area should look like the following image:



Make the employee ID field read-only

The employee ID field is disabled because the editable property on the field is set to false. However, the default behavior of the text field binder changes the enabled state on the field when the data object contains a value. You can turn off this binder behavior so that the field will remain in its initial read-only state.

To prevent the binder from automatically switching the editable property:

1. Select the Employee ID field. A line displays on the design area with an icon representing the binder for the field.
2. Click the binder icon for the Employee ID field.
3. In the Properties view, change the autoEditable property to **false**. Press **Enter**.

Lesson checkpoint

Now, when you run the application and select an employee from the table, the details of that employee's record are displayed in the details fields.

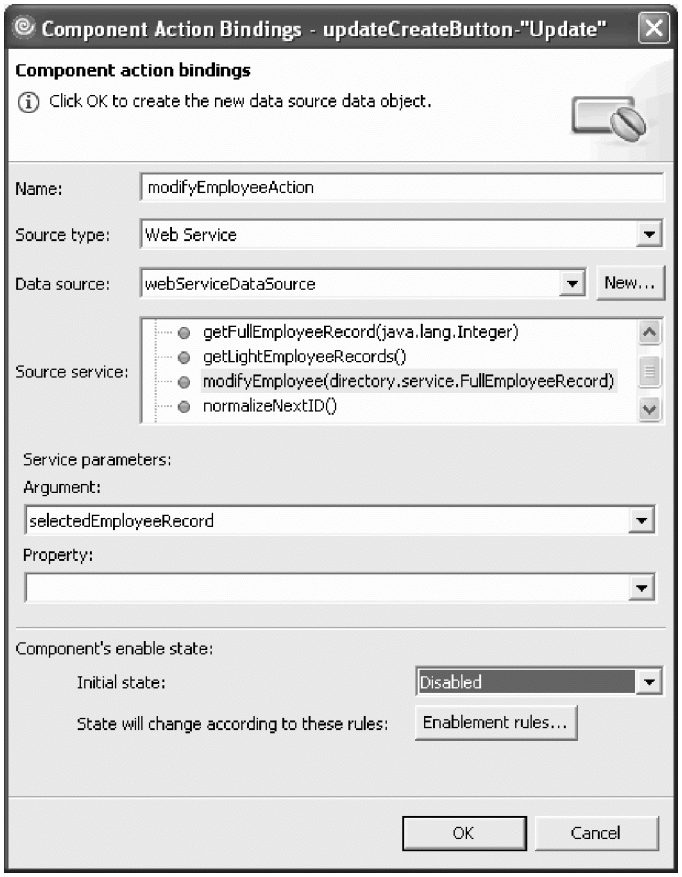
Lesson 2.4: Bind the Update button to an action binder

The Java visual editor provides action binders for calling a service on a data source when a button is clicked. For example, when the Update button is clicked, the application should run a modifyEmployee() method on the Web service with the changes entered into the details fields. In this lesson, you will bind the Update button to an action binder.

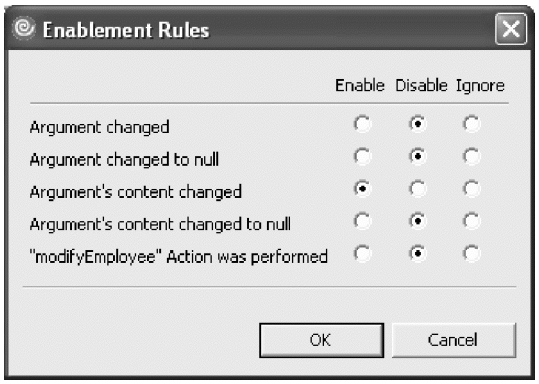
To bind the Update button:

1. Select the **Update** button in the design area, and click the **Bind** tab to open the Component Action Bindings dialog box.

2. In the **Source type** field, select **Web Service**.
3. In the **Data source** field, select **webServiceDataSource**.
4. From the **Source service** list, select **modifyEmployee(directory.service.FullEmployeeRecord)**.
5. The **Name** field automatically changes to **modifyEmployeeAction**. Accept this default.
6. In the **Argument** field, select **selectedEmployeeRecord**.
7. Because the modifyEmployee() method takes a full employee record as its argument, you must leave the **Property** field blank.
8. Set the **Initial state** of the button to **Disabled**.



9. To define how the button changes its state, click **Enablement rules**. Specify that the button is enabled only when the argument's content is changed, and disabled in all other instances. Click **OK**.



   This means that the **Update** button is disabled until the contents of the selectedEmployeeRecord changes. In other words, as soon as you type a new value in one of the details fields, which are bound to the selectedEmployeeRecord, the binder enables the button. If you select a new record or click **Update**, the button will become disabled again.

10. Click **OK**.

A new SwingDataServiceAction binder is added for the **Update** button. If you select the button in the design area, the visual editor draws a line that indicates that the button is bound to the Web service data source. A pink, dotted arrow points from the selectedEmployeeRecord object to the line. This arrow indicates that the selectedEmployeeRecord is the argument for the call to the service.

Lesson checkpoint

Now, when you run the application, you can update an employee's record.

Select an employee in the table and change the last name. As soon as you change the last name, the **Update** button is enabled. When you click **Update**, the modifyEmployee service is called and the employee is updated. The new last name is reflected in the employees table.

Lesson 2.5: Enable the Delete button and confirmation dialog box

In this exercise, you will program the My Company Directory application to delete an employee record.

The following list describes the behavior that you want the application to use:

- When you select an employee in the table, the **Delete** button is enabled.
- When you click the **Delete** button, the Confirm Delete dialog box opens and asks you to confirm the deletion.
- If you click the **Yes** button on the Confirm Delete dialog box, the employee record is deleted, the Confirm Delete dialog box closes, and the list of employees is refreshed.
- If you click **No**, the deletion is canceled and the Confirm Delete dialog box closes.

Program the Delete button to be enabled or disabled based on whether a row is selected in the table

To program the Delete button to be enabled or disabled, add a listener to the table that enables the button when a row is selected.

1. Select the employeesTable in the Java Beans view. The source view highlights the following line:

   ```
   employeesTable = new JTable();
   ```

2. Immediately after this line, add a new ListSelectionListener and valueChanged event to the employeesTable:

   ```
   employeesTable.getSelectionModel().addListSelectionListener(new ListSelectionListener() {
      public void valueChanged(ListSelectionEvent e) {
        getDeleteButton().setEnabled(getEmployeesTable().getSelectedRowCount() != 0);
      }
   });
   ```

3. After you add these lines of code, the source editor marks them as errors until you import ListSelectListener and ListSelectionEvent. To add the required imports, click **Source** → **Organize Imports** on the main menu. The following lines are added to the imports section of the class:

   ```
   import javax.swing.event.ListSelectionEvent;
   import javax.swing.event.ListSelectionListener;
   ```

Now, when a row in the table is selected, the **Delete** button is enabled.

Program the Confirm Delete dialog box to open when Delete is clicked

Add an actionPerformed event to the Delete button, and program the event to open the Confirm Delete dialog box.

1. Right-click the **Delete** button and select **Events** → **actionPerformed**. The following event stub is added to the getDeleteButton() method:

   ```
   deleteButton.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent e) {
        System.out.println("actionPerformed()");
        // TODO Auto-generated Event stub actionPerformed()
      }
   });
   ```

2. Replace this generated stub with the following code, which sets the Confirm Delete dialog box to be visible when the button is clicked:

   ```
   deleteButton.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent e) {
        getConfirmDialog().setVisible(true);
      }
   });
   ```

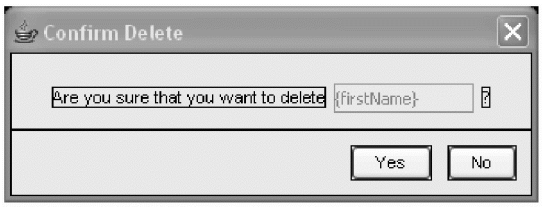
Bind the text field in the Confirm Delete dialog box

Bind the text field in the Confirm Delete dialog box to display the first name of the employee to be deleted.

1. On the Java Beans view or design area, select the employeeToDeleteField text field, and click the **Bind** tab.

2. On the Field Data Bindings dialog box, select the selectedEmployeeRecord data object and the firstName field, then click **OK**.

   The text field is now bound to the firstName column of the selected row in the employeesTable.



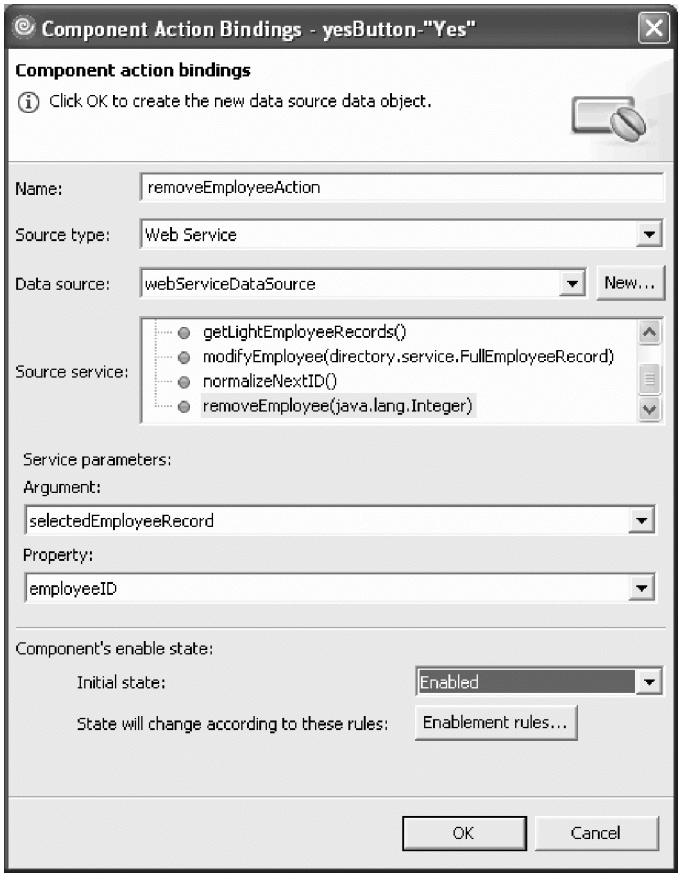
3. To make sure that this field is read-only, set the **autoEditable** property for the field's binder to **false**.

Bind the Yes button to perform the deletion

Bind the **Yes** button to call the removeEmployee(java.lang.Integer) method on the Web service.

1. Select the **Yes** button, and click the **Bind** tab to open the Component Action Bindings dialog.
2. In the **Source type** field, select **Web Service**.
3. In the **Data source** field, select **webServiceDataSource**.
4. From the **Source service** list, select **removeEmployee(java.lang.Integer)**.
5. The **Name** field automatically changes to **removeEmployeeAction**. Accept this default.
6. In the **Argument** field, select **selectedEmployeeRecord**.
7. In the **Property** field, select **employeeID**. Because the removeEmployee() method takes an integer as its argument, you use the employee ID of the selectedEmployeeRecord.
8. Set the **Initial state** of the button to **Enabled**.
9. For the **Enablement rules**, select **Ignore** for each of the conditions.

   This component state means that the Yes button will always be enabled, since there is no need for it to change its state.



10. Click **OK**.

Add an event to hide the Confirm Delete dialog after the employee is deleted

In this step you add an event to the **Yes** button's *binder* (not the **Yes** button itself). You want the Confirm Delete dialog box to close after the employee is removed, which means after the binder has successfully called the service on the data source.

Add the following code to the getRemoveEmployeeAction() method:

```
removeEmployeeAction.addActionBinderListener(new jve.generated.IActionBinder.ActionBinderListener() {
   public void afterActionPerformed(jve.generated.IActionBinder.ActionBinderEvent e) {
     getConfirmDialog().setVisible(false);
   }
   public void beforeActionPerformed(jve.generated.IActionBinder.ActionBinderEvent e) {}
});
```

This event code hides the Confirm Delete dialog box after the binder's action is performed.

Lesson checkpoint

Now, when you run the My Company Directory application you can select an employee in the table, click the **Delete** button, and click **Yes** to confirm the deletion. The employee record will be removed from the directory, and the list of employees will reflect the removal.

Lesson 2.6: Set up actions and bindings for adding a new employee

In this lesson you enable the My Company Directory application to add a new employee record.

Because the behavior of the application is more complicated and dynamic for adding a new employee, this exercise is inherently more complex and requires you to make some manual changes to the source code. Also, this exercise demonstrates some advanced capabilities of the data objects, and it gives you a creative example for ways that you can use the binders and data objects to fit your needs.

The following list describes the required behavior of the application:

- When you click the **New** button, the following behavior occurs:
  - The selection is cleared on the employees table, and the table is disabled.
  - Clearing the table selection causes the **Delete** button to be disabled.
  - The **Filter** field is disabled.
  - The details fields are cleared of any values, except for a new employee ID.
  - The text on the **Update** button switches to **Add**.
- When you click the **Add** button, the following behavior occurs:
  - The values entered into the details fields are added to the directory as a new employee record.
  - The table is enabled and the values are refreshed.
  - The **Filter** field is enabled.
  - The text on the **Add** button switches back to **Update**.

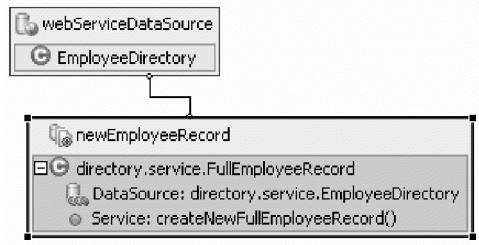
Add a new Data Source Data Object that calls createNewFullEmployeeRecord()

The sample Web service provides a createNewFullEmployeeRecord service that provides a new, blank employee record that is populated with the next available employee ID number. This blank record can then be populated with a new employee's information and submitted back to the Web service.

1. On the palette of the Java visual editor, expand the Data Objects drawer and select **Data Source Data Object**.
2. Move your mouse pointer over the blank area of the design view, or free-form area, and left-click to drop the Data Source Data Object. A new Data Source Data Object is added and shown on the free-form area:

3. Right-click the Data Source Data Object, and select **Rename field**. Rename the data object to `newEmployeeRecord`.
4. Right-click the newEmployeeRecord data object, and select **Binding Properties**. The Data Binding dialog box opens.
5. In the **Data source** field, select webServiceDataSource
6. In the **Service** field, select createNewFullEmployeeRecord()
7. Click **OK**.

On the free-form area, you can see that the newEmployeeRecord data source data object is bound to the Web service.

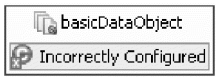


Add a Basic Data Object to facilitate the switching of data objects

Because the details fields and the Update button need to switch modes (for both performing an update and creating a new employee), they need to be bound to two different data objects at different times. To facilitate this step, you will add a Basic Data Object named switchingDataObject. You will use this Basic Data Object to switch the binding for the text fields between the selectedEmployeeRecord and the newEmployeeRecord.

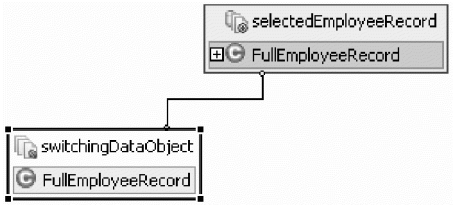
The new Basic Data Object simply points to another data object (selectedEmployeeRecord) that you defined in an earlier exercise. This new data object will become useful when you create a method that tells this basic data object to use the newEmployeeRecord that you created earlier. In other words, this basic data object will function as an intermediate data object that switches between the selectedEmployeeRecord data object and the newEmployeeRecord data object, allowing visual components in your application to work with two different data objects.

1. On the visual editor palette, select **Basic Data Object**, and drop it onto the free-form area. A basicDataObject is added.



2. Rename the data object to `switchingDataObject`
3. In the Properties view for switchingDataObject, set the **sourceObject** property to **selectedEmployeeRecord**. You can select selectedEmployeeRecord from the drop-down menu in the Value column for the property.

   Now, switchingDataObject refers to selectedEmployeeRecord and reflects the same values:

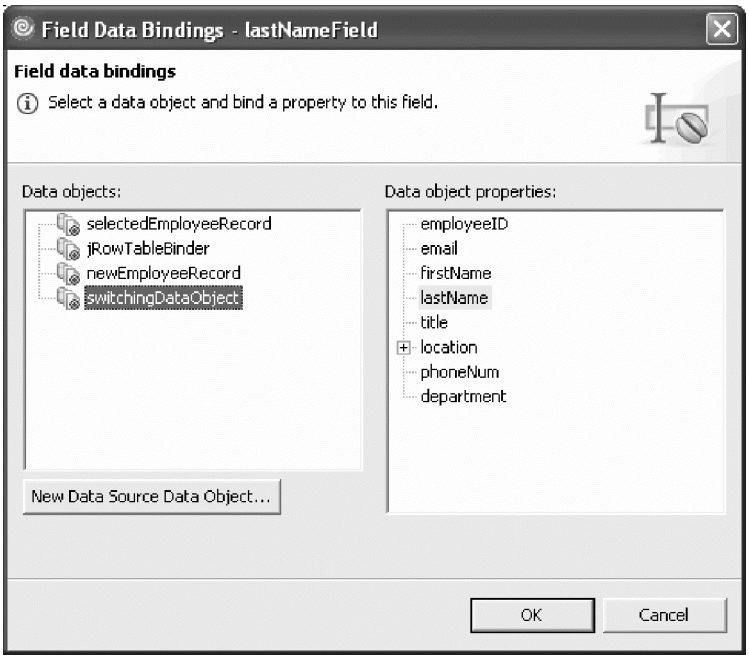


Rebind each employee field to the switchingDataObject

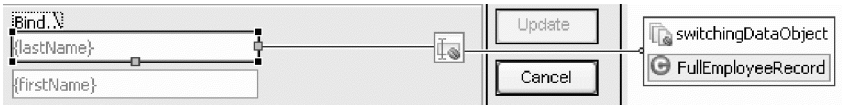
Even though each of the employee details fields is already bound to selectedEmployeeRecord, you will now bind them to switchingDataObject. After binding the fields, you can dynamically switch between data objects for the fields, depending on whether you are modifying an existing employee record or adding a new employee record.

For each of the fields in the Employee details section, complete the following steps:

1. Select the field and click the **Bind** tab.
2. On the Field Data Bindings dialog box, select the switchingDataObject. You previously bound the fields to the selectedEmployeeRecord.



3. Make sure the field is still bound to the correct data object property, and click **OK**. If you select the field on the design view, you can see that the binder lines now point to switchingDataObject.



Define a flag and a method for updating and switching modes

The following updateMode() method checks to see if the mode flag is set to new, then changes the application's behavior accordingly. By default, the Boolean flag isNewMode is set to false, and the updateMode() method enables the employees table and the filter field, and sets the text on the Update button to "Update". If isNewMode is set to true, the employees table is disabled and cleared of any selection, the filter field is disabled, and the text on the Update button is set to "Add".

Add the following code to your DirectoryApp.java class just before the last closing curly brace:

```
private boolean isNewMode = false;
private void updateMode() {
   if (isNewMode) {
      getEmployeesTable().clearSelection();
      getEmployeesTable().setEnabled(false);
      getFilterField().setEditable(false);
      getUpdateCreateButton().setText("Add");
   } else {
      getEmployeesTable().setEnabled(true);
      getFilterField().setEditable(true);
      getUpdateCreateButton().setText("Update");
   }
}
```

Add an actionPerformed event to the New button

In this step, you add event code for when the **New** button is clicked. The event tells the switchingDataObject to use the newEmployeeRecord data object, sets the mode flag to "new," and runs the updateMode() method that you added in the previous step.

1. In the design view, right-click the **New** button, and select **Events** → **actionPerformed**. The following code is generated in the getNewButton() method:

   ```
   newButton.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent e) {
         System.out.println("actionPerformed()"); // TODO Auto-generated Event stub actionPerformed()
      }
   });
   ```

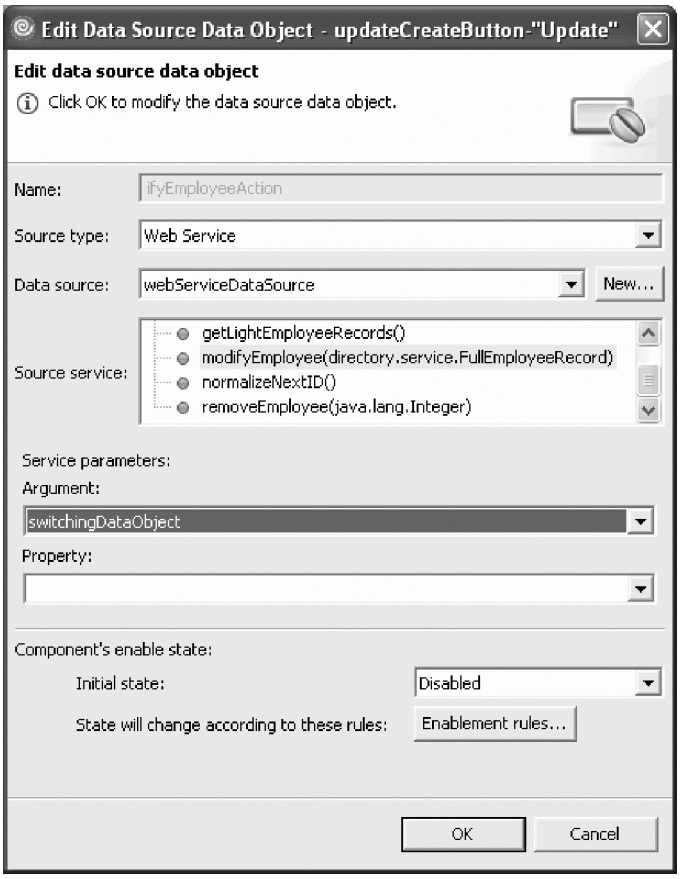
2. Replace this generated stub with the following code:

   ```
   newButton.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent e) {
         getSwitchingDataObject().setSourceObject(getNewEmployeeRecord());
         getNewEmployeeRecord().refresh();
         isNewMode = true; //sets application to new mode
         updateMode(); //changes UI according to new mode
         getLastNameField().grabFocus();
      }
   });
   ```

Rebind the Update button

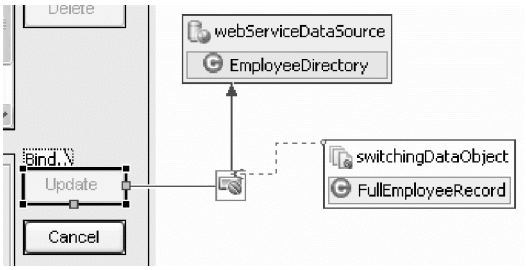
In a previous lesson, you programmed the **Update** button to use the modifyEmployee method on the Web service. That action is implemented as a SwingDataServiceAction. One of the properties of the SwingDataServiceAction is the source object, which acts as the argument for the service. The source object for the modify action is currently set to selectedEmployeeRecord. In order to program the button to control both an update and an addition, you will reconfigure the button's action to use switchingDataObject as an argument to the modifyEmployee service.

1. In the design view, select the **Update** button. Notice the pink, dotted arrow showing that the selectedEmployeeRecord is the argument for the service call.
2. Click the **Bind** tab on the **Update** button.
3. In the **Argument** field, select switchingDataObject.



4. Click **OK**.

   Now, notice that the button's action is now configured to use the switchingDataObject as its argument to the modifyEmployee method:



Add an event to the Update button's binder to reset the mode

After the **Update** button is clicked and the action is complete on the Web service, you want the application to go back into its default mode and behavior. To do this, you add an event listener on the button's action binder that will update the mode and refresh the table after the update or addition is performed.

Add the following code to the getModifyEmployeeAction() method for the Update button:

```
modifyEmployeeAction.addActionBinderListener(
  new jve.generated.IActionBinder.ActionBinderListener() {
    public void afterActionPerformed(jve.generated.IActionBinder.ActionBinderEvent e) {
      if (isNewMode) {
```

```
        //Go back to using the selectedEmployeeRecord
        getSwitchingDataObject().setSourceObject(getSelectedEmployeeRecord());
        //Revert out of new mode
        isNewMode = false;
        updateMode();
      }
      // Refresh the table's data object
      getLightEmployeeRecordRows().refresh();
    }
    public void beforeActionPerformed(jve.generated.IActionBinder.ActionBinderEvent e) {}
});
```

Lesson checkpoint

Now, when you run the My Company Directory application, you can click the **New** button and add a new employee record.

Lesson 2.7: Program the Cancel button behavior

When using your application, you want to be able to easily back out of any changes that you start to make to an employees record if you decide not to submit the changes. In other words, you need to be able to cancel and clear the fields so you can start over. To add this functionality, you set some actionPerformed events on the **Cancel** button.

The following list describes the required behavior of the **Cancel** button:

- If you click the **Cancel** button while in new mode, the application reverts out of new mode.
- If you click the **Cancel** button while modifying an employee record, any values that you have changed revert back to the original values.

To add an actionPerformed event to the **Cancel** button to perform the required behavior:

1. In the design view, right-click the **Cancel** button, and select **Events → actionPerformed**. The following code is generated in the getCancelButton() method:

```
cancelButton.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent e) {
      System.out.println("actionPerformed()"); // TODO Auto-generated Event stub actionPerformed()
   }
});
```

2. Replace the generated event stub with the following code:

```
cancelButton.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent e) {
      if (isNewMode) {
         getSwitchingDataObject().setSourceObject(getSelectedEmployeeRecord());
         isNewMode = false;
         updateMode();
      } else {
         getSelectedEmployeeRecord().refresh();
      }
   }
});
```

Lesson checkpoint

In this lesson, you learned how to program the **Cancel** button with actionPerformed events.

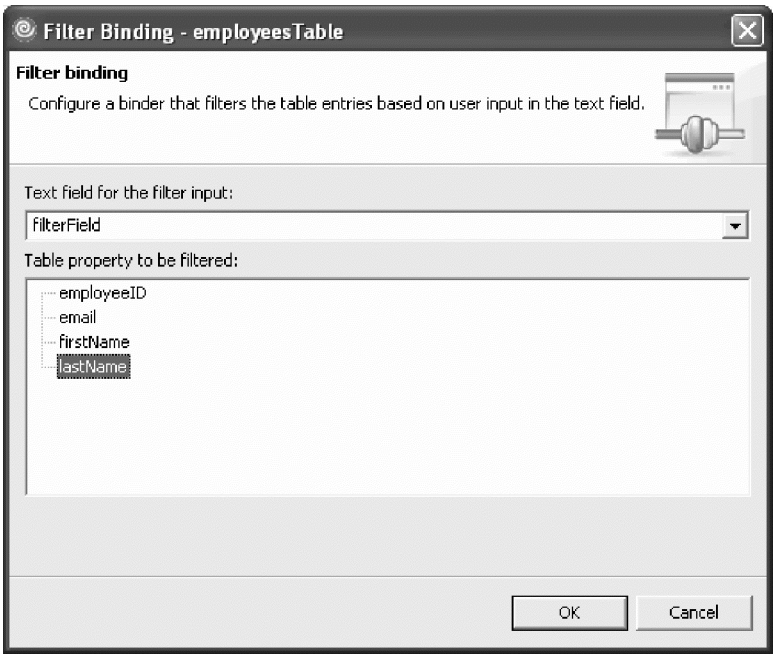
Lesson 2.8: Set up a filter on the employees table

You can use a Text Filter Binder to filter the contents of the employees table. The filter takes input from a text field, and filters the table based on a particular property, or column, in the table.

In the application, you will use the characters entered in the **Filter** field to filter by employee last name. If the exact values entered in the **Filter** field are present in the last name of an employee record, the employee record will display in the table.

To create a filter for the table:

1. Select the binder icon for the employeesTable and select **Filter Binding Properties**. The Filter Binding dialog box opens.
2. In the **Text field for the filter input** list, select **filterField**.
3. In the **Table property to be filtered** list, select **lastName**.



4. Click **OK**.

A new SwingPropertyFilter is generated. The filter property on the table's binder is set to use the new filter. The new filter is configured to use the **Filter** field for its input, and to filter on the lastName property of the table.

Lesson checkpoint

In this lesson you learned how to set up a filter for a table.

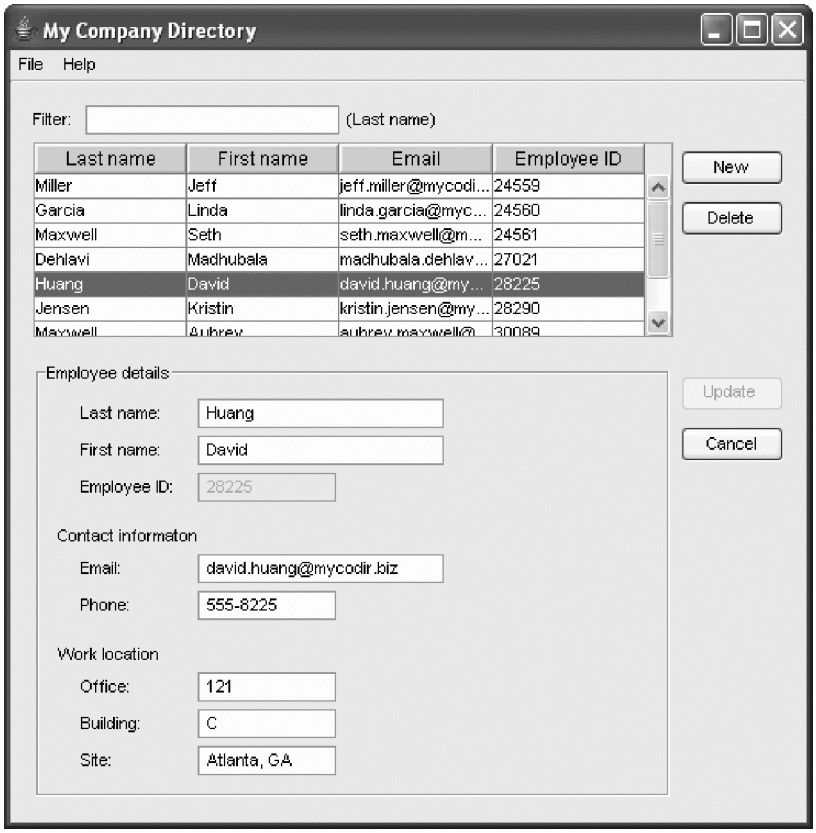
Now, when you run the My Company Directory application, you can type characters in the **Filter** field, and the table will be filtered to show the rows where the last name contains the characters entered.

Congratulations! The My Company Directory application is finished.

Summary: Build a rich Java client that uses a Web service

Congratulations! You learned how to use the Java visual editor to build the My Company Directory application, a rich Java client that connects to a sample Web service to maintain an employee directory.

See a picture of the finished product:



Lessons learned

You used the visual editor to complete the graphical user interface, using GridBagLayout to arrange the employees table. Then you bound the table, fields, and buttons to appropriate data objects and data source to make the application work with a Web service Java proxy that you generated. You also did some complex coding to make the application behave properly to make it easy to use and intuitive. And, you learned how to install an enterprise application on WebSphere Application Server v6.0 and deploy a Web service.

Most importantly, you learned all about the powerful binder class provided by the Java visual editor for working with data. Now you are ready to begin experimenting on your own and putting the binders to new and exciting uses.

You should now be able to do the following tasks:

- Use the Java visual editor to layout components in a GridBagLayout.
- Run a visual class as a Java bean.
- Bind a Java application's visual components to methods and data objects returned by a Web service.
- Add events to visual components.

Additional resources

Import a finished version of the My Directory application

This project includes the finished application, the jve.generated package with binder classes, and the Web service Java client configured for WebSphere Application Server v6.1. If you import this finished project without working through the tutorial, you may need to configure your Java build path variable. It needs to point to your WebSphere v6.1 Web services thin client JAR file.

Tip: Unless you specify a different project name during import, this will overwrite your MyDirectory project contents.