Connecting a GUI to a Java DB Database with NetBeans IDE

Contributed by Talle Mulligan and maintained by Patrick Keegan

This tutorial guides you through the process of connecting the GUI of an application called ContactEditor to a Java DB database. In the process you will add data-aware JDBC components to the GUI that will enable the program to interact with an employee database.

In this tutorial you will learn how to:

- Use the GUI Builder interface
- Connect two GUI windows
- Add and edit a `JDBCRowSet`
- Add a data model
- Connect a GUI to a Java DB database
- Bind data to UI components
- Connect UI components to application logic

This tutorial takes about 30 minutes to complete.

Note that this tutorial refers to a sample project called GUI DB Exercise Initial which you can download and use while working through the steps. You can also download the GUI DB Exercise Final project showing the completed application.

Note: In order to complete this tutorial successfully, you must be running NetBeans IDE 5.5 on JDK 5.

Getting Started

In the preceeding Working with the Java DB (Derby) Database in NetBeans 5.5 tutorial, we concentrated on setting up Java DB in NetBeans, creating a Contacts table in the sample database in the process. In this tutorial, we'll connect a detail dialog and the main (parent) window of a Java GUI form like the one we created for the `ContactEditorUI.java` application in the GUI Builder Quickstart to a Java DB database.

Note that for NetBeans IDE 5.5, the included database is no longer called Derby and is referred to as Java DB. Though this tutorial's included images illustrate the process on Macintosh OS-X, the steps are virtually identical for other supported platforms, such as Windows or Solaris.

If you're continuing from the previous Working with the Java DB (Derby) Database in NetBeans 5.5 tutorial you should be ready to bind the database to the UI included in the GUI_DB_Exercise_Initial sample project. Before proceeding, however, ensure that the Java DB engine is running and if not start it up now by choosing Tools > Start Java DB Server. Also, make sure that you are connected to the contact_data database by right-clicking the database connection node in the Runtime window and choosing Connect. Once Java DB has been started, reconnect to the contact data database using nbuser for both the User Name and Password.

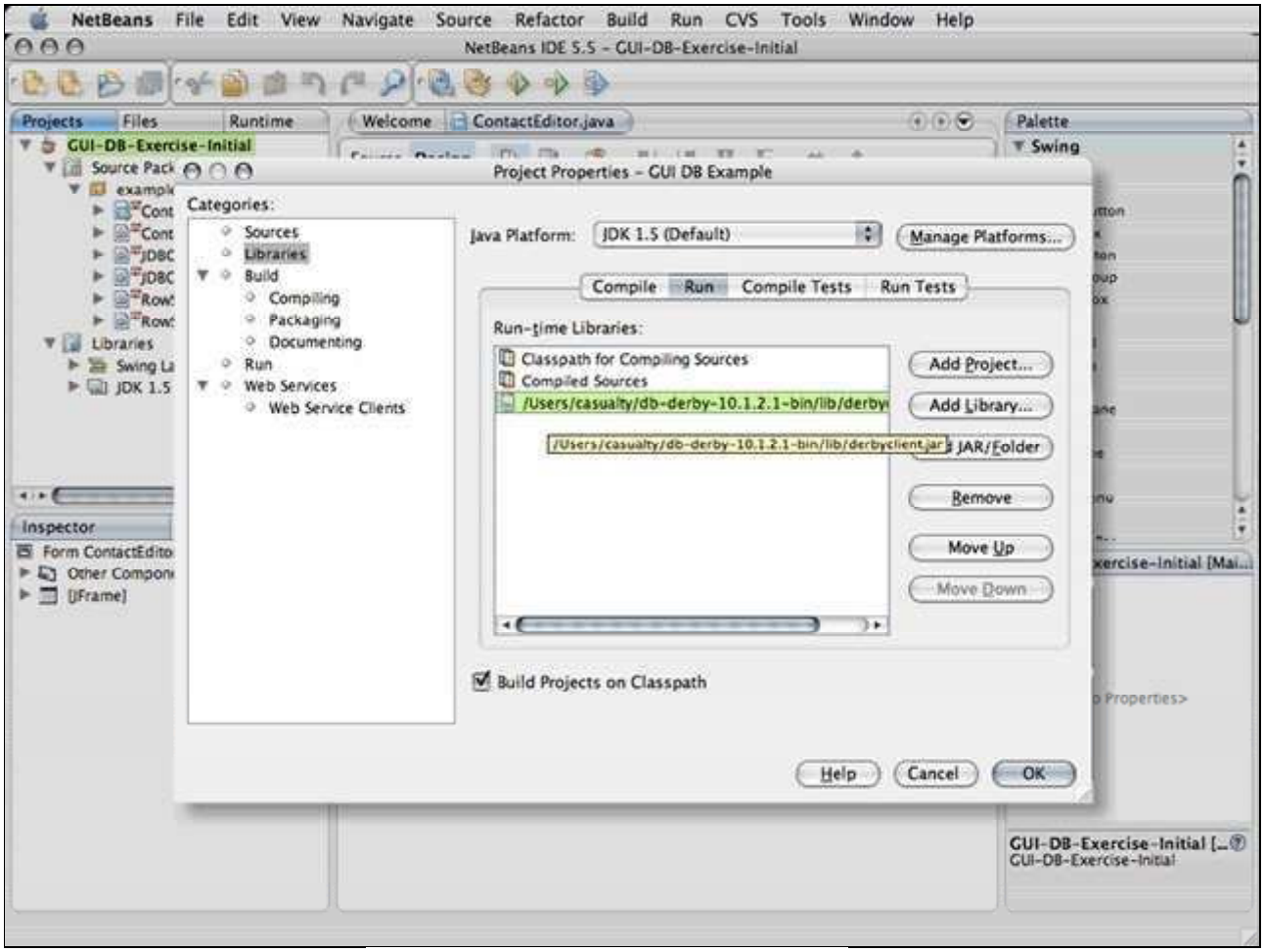
Adding the Java DB Client Library to the Project Classpath

Once you've opened the `GUI_DB_Exercise_Initial` sample project, you should see the `ContactEditor.java` form along with a few additional support classes in the Projects window. Note that for the purposes of the tutorial, both the necessary support classes and the ContactEditor dialog UI we built in the first Matisse tutorial have already been included the project so we can focus on the business of binding our database data to the GUI.

One thing we do have to do, however, is to add the Java DB client library JAR that is located in the Java DB installation directory to our project's classpath. We then need to Build the project to compile everything so that we'll be able to copy and paste the necessary components into the form.

To add the `derbyclient.jar` library to the classpath:

1. In the Projects window, right-click the project node and choose Properties from the pop-up menu.
2. In the project Properties window that appears, select the Libraries node in the Categories pane. Then select the Run tab.
3. Click the Add JAR/Folder button, navigate to the location of your Java DB installation.
4. Select `derbyclient.jar` and click Open to add the JAR to the project classpath.
5. Click OK to exit the dialog.



Adding the Java DB Client JAR to the Project

Preparing the Support Classes

In order to use the support classes provided for this tutorial, you need to compile them first. Right-click the project node and choosing Build Project from the pop-up menu. If all goes well, the Output window will return the BUILD SUCCESSFUL message and we'll be able to move on.

The required support classes are as follows:

- `ContactsModel.java`. A class which provides the logic for the interaction with the database (including reading and writing data). The code here is specific to the database table you create in the tutorial, but you can use much of the code as a template for your own app.
- `JDBCRowSet.java`. A general wrapper class for `com.sun.sql.JDBCRowSet`, which enables your application to interact with a database.
- `JDBCRowSetBeanInfo.java`.
- `RowSetTableModel.java`. A custom table model that is used by the JTable in the application to display columns and rows according to the . This table model takes information provided by the JDBCRowSet object to populate the table in the application.
- `RowSetTableModelBeanInfo.java`.

Used together, these classes make it possible to bind data to Swing JTable cells.

`ContactsModel.java`, `JDBCRowSet`, and `RowSetTableModel` are beans. In other words, the are written to the JavaBeans specification, which enables you to use those classes much as you would use other components in the GUI Builder. For example, you can use drag and drop to add these beans to a form. Then you can use the property sheet for these components to set properties (instead of manually coding the properties).

`RowSetTableModelBeanInfo`and `JDBCRowSetBeanInfo` are BeanInfo classes. The IDE uses the BeanInfo classes to determine which properties for a bean to show in the GUI Editor.

If you want to use JDBC to bind databases to Swing components for your own apps, you can add `JDBCRowSet` and `RowSetTableModel` to the component palette and make use of them similar to how the are used in this tutorial.

Preparing the CONTACTS Table

We will be working with the CONTACTS table that we added to the database in the previous tutorial. If you need to recreate this table now, you can do so by executing the following SQL statement in the SQL Editor:

```
CREATE TABLE "CONTACTS"
(
"ID" INTEGER not null primary key,
"FIRST_NAME" VARCHAR(50),
"LAST_NAME" VARCHAR(50),
"TITLE" VARCHAR(50),
"NICKNAME" VARCHAR(50),
"DISPLAY_FORMAT" SMALLINT,
"MAIL_FORMAT" SMALLINT,
"EMAIL_ADDRESSES" VARCHAR(1000)
);
```

Note that when using the SQL Editor you may need to refresh the Runtime window to display the new Contacts table node. You can do this by right-clicking the connection node and choosing Refresh from the pop-up menu. If you are using the SQL Editor, you might need to refresh the Tables node in the Runtime window in order to see the Contacts table.

top

Adding the Data Model

Since the application's main window GUI has already been laid out for us, we can jump straight to adding the data-aware components that will enable us to interact with our data in the database. In this section, we'll add the data model that will ensure the data is presented to the UI in the correct form.

Adding a JDBCRowSet

Because the JDBCRowSet we require is already included in the GUI DB Example project, we only need to add it to our application GUI. Since the included RowSets have BeanInfos, we can use them as we would other components with the IDE displaying their properties in the Properties window. Because we're only going to use this class once, we don't need to bother installing the JDBCRowSet into the Palette in order to add it to the form. Though this approach is worthwhile if you want to use a class repeatedly, we will simply copy and paste to save time.

To add a JDBCRowSet to the form:

1. In the Projects window, right-click the `JDBCRowSet.java` node and choose Copy from the pop-up menu.
2. Paste it anywhere in the Design Area.

   The IDE adds the JDBCRowSet to the form and a node representing the row set appears in the Inspector window.

Notice that the Inspector window node is added within the Other Components node. If you receive an error when attempting to paste the row set, compile the project and then try again.

Setting the JDBCRowSet's Properties

Now we need to edit the JDBCRowSet properties so that it refers to the Contacts database we created earlier. We also need to set the driver and path it should use, as well as provide the password and username it will need to make the connection.

To edit the JDBCRowSet's properties:

1. Select the JDBCRowSet1 node in the Inspector window (not the Projects window).
2. In the Properties window, enter `select * from contacts` for the command property.
3. Enter `org.apache.derby.jdbc.ClientDriver` for the driver property.
4. Enter `jdbc:derby://localhost:1527/contact_database` for the url property.
5. Enter `nbuser` for the password property.
6. Enter `nbuser` for the username property.

Adding the Data Model

Now we need to add the data model that will be the layer between the application and our database as well as encapsulate data access, providing the logic for accessing and modifying the data itself. Once again, the necessary `ContactsModel` class has been provided in the example project so we'll need only add it to our application. Including such a class enables the possibility of changing the data model without necessitating changes to the GUI.

To add the data model to the form:

1. In the Projects window, right-click the `ContactsModel.java` node and choose Copy from the pop-up menu.
2. Paste it anywhere in the Design Area.

The IDE adds the ContactModel to the form and a node representing the row set appears within the Other Components node in the Inspector window.

To change the variable name of the data model:

1. Right-click the contactsModel1 node in the Inspector window and choose Change Variable Name from the pop-up menu.
2. In the Rename dialog that appears, enter contactsModel for the new variable name and click OK.

To set the rowSet property of the data model:

1. Select the contactsModel node in the Inspector window.
2. In the Properties window, click the ellipsis button (...) for the rowSet propert .
3. In the editor that appears, select the Bean radiobutton in the Get Parameter From pane.
4. Select jDBCRowSet1 from in the Select Bean combobox.
5. Click OK to exit the dialog.

   The IDE sets the form's contactsModel data model to use the jDBCRowSet.

top

Binding the Database to the UI

In order for our GUI to interact with the contact information stored in our database, it's necessar to bind the data to the components that will displa that data. In this section, we'll start connecting our GUI to the JDBC components that will enable interaction with the database.

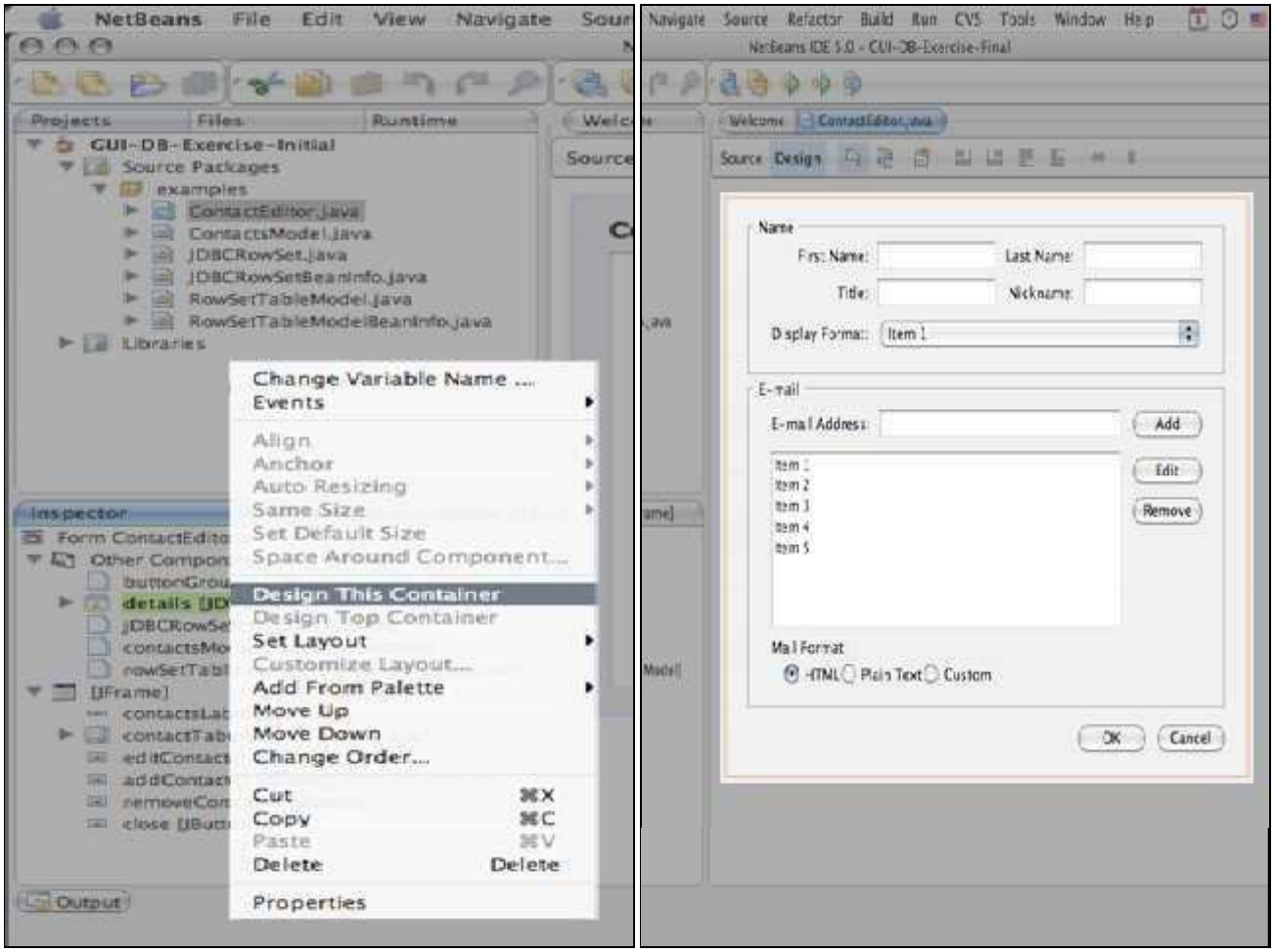
Changing GUI Builder Focus

Though we've alread completed the necessar modifications to the details dialog, we need to shift the GUI Builder's focus back to the top-level Contacts container that will be our application's main window before we can begin binding the database to our GUI.

To change the GUI Builder's focus:

1. In the Inspector window, right-click the details dialog's node and choose Design Top Container from the pop-up menu.

   The IDE shifts focus to the Contacts JFrame container.



Changing GUI Builder Focus, Before and After

Setting the JTable RowSet Model

In order for our GUI's JTable to displa the contact data properl , we need to set it to use the provided RowSetTableModel class. We also need to point the RowSetTableModel to the JDBCRowSet we added earlier so that it will be able to forward column information and data to the JTable itself. Finall , we'll edit the rowSetTableModel1 instance so that onl those columns which we want to displa will be visible in the GUI's main window.

To add a RowSetTableModel to the form:

1. In the Projects window, right-click the `RowSetTableModel.java` node and choose Cop from the pop-up menu.
2. Paste it an where in the Design Area.

To set the rowSet property of the table model:

1. In the Inspector, select the rowSetTableModel1 node.
2. In the Properties window, open the rowSet propert editor b clicking its ellipsis button (...).
3. Select the Bean radiobutton in the Get Parameter From pane.
4. Select `jDBCRowSet1` from in the Select Bean combobox.
5. Click OK to exit the dialog.

To set the model property of the JTable:

1. In the Inspector window, expand the contactTablePane[JScrollPane] node and select the contactTable[JTable] node. Note that when ou select the JTable in the form, the JScrollPane component is highlighted in the Inspector and its properties are available for editing in the Properties window (not the JTable's properties).
2. In the Properties window, open the editor for the model propert b clicking its ellipsis button (...).
3. In the model editor that appears, choose Form Connection in the Select Mode combobox.
4. Select the Bean radiobutton and choose `rowSetTableModel1` in the Component combobox.
5.
6. Click OK to close the editor.

In order to determine which table columns will be visible at runtime, we need to set them explicitly. In order to do this, we need to adjust the rowSetTableModel1's properties.

To set the visibility of the table columns:

1. In the Inspector, select the rowSetTableModel1 node.
2. In the Properties window, open the visibleColumns property editor by clicking its ellipsis button (...).
3. In the editor that appears, enter the following column names in the Item field one at a time and click Add.
    - NICKNAME
    - FIRST_NAME
    - LAST_NAME
4. Click OK to exit the dialog.

   The IDE sets the form's JTable to display the specified columns and the column titles appear in the form itself in the order in which they were added.

Setting Main Window Close Behavior

In order to release the various resources held in the model when application windows are closed, we also need to specify the desired behavior explicitly. In this section we'll set the event actions that control this behavior for both the application's main window and the details dialog.

To edit event actions for the main window Close button:

1. Right-click the Close button and choose Events > Action > actionPerformed.
2. In the Source Editor, select the line where the cursor is located (it should read //TODO add your handling code here) add the following code in the body of the closeActionPerformed method:

```
contactsModel.dispose(); // releases resources held by the model (like DB connection)
System.exit(0);  // exits the application
```

3. Click the Design button to return to the GUI Builder.

To edit event actions for the main window:

1. Right-click the JFrame container in the Inspector and choose Events > Window >windowClosing.
2. In the Source Editor, select the line where the cursor is located. Then add the following code in the body of the formWindowClosing method:

```
contactsModel.dispose(); // releases resources held by the model (like DB connection)
System.exit(0);  // exits the application
```

3. Again, click the Design button to return to the GUI Builder.

Adding Button Event Handlers

In order for the main window's various buttons to result in the desired behavior, the actions have to be set on them. In this section we'll do just that by defining the event handlers necessary for them to trigger the appropriate events.

To add an event handler to the Edit button:

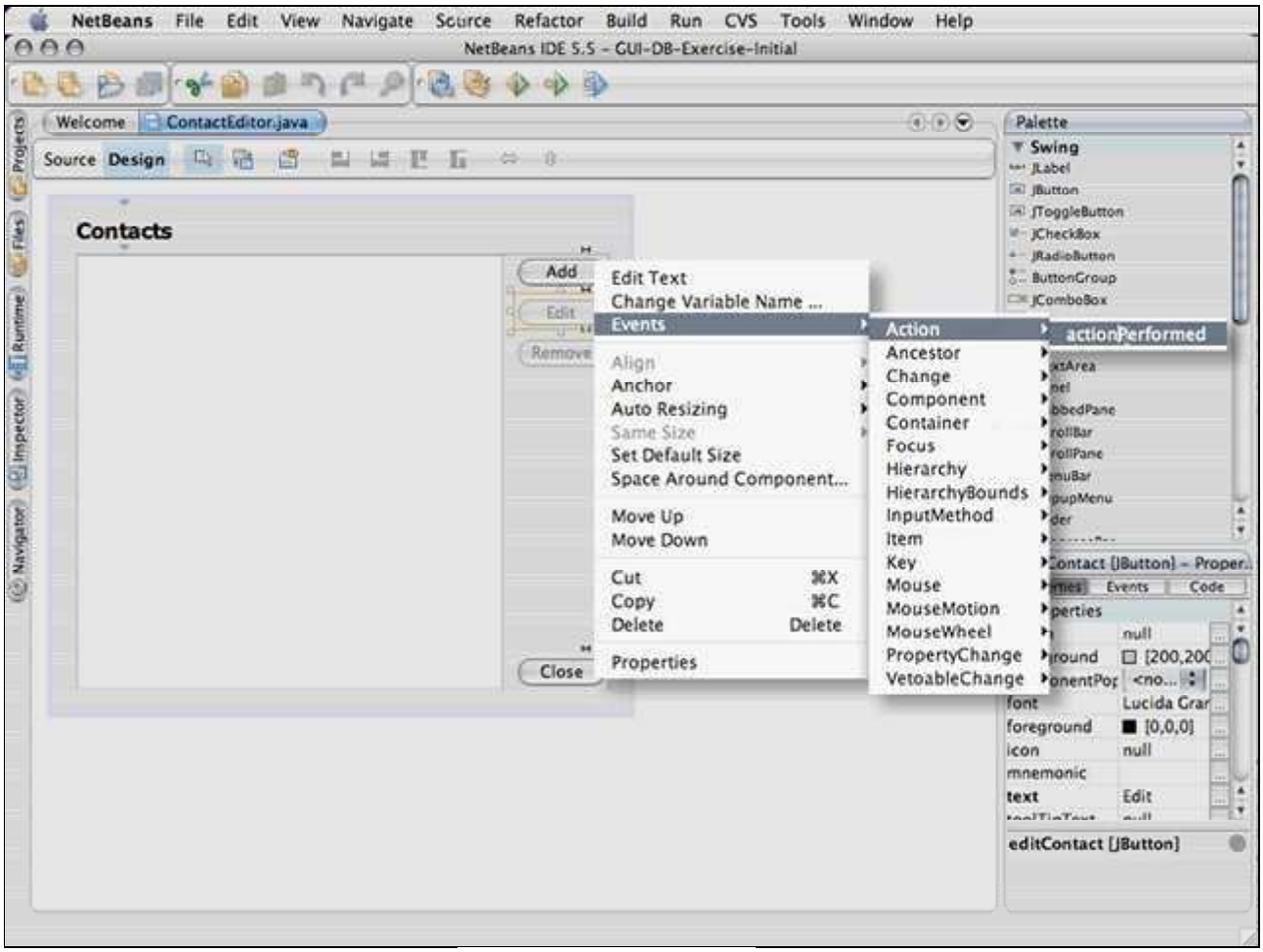
1. Right-click the Edit button and choose Events > Action > actionPerformed as shown in the figure below these steps.
2. In the Source Editor, select the line where the cursor is located and add the following code in the body of the editContactActionPerformed method:

```
showDetailDialog(false);
```

3. Add the following method into the code in the body of the contactEditor class immediately below the closeActionPerformed method (on or about line 346).

```
private void showDetailDialog(boolean inserting) {
    contactsModel.setInsertMode(inserting);
    firstNameField.setText(contactsModel.getFirstName());
    lastNameField.setText(contactsModel.getLastName());
    titleField.setText(contactsModel.getTitle());
    nicknameField.setText(contactsModel.getNickname());
    displayFormat.setSelectedIndex(contactsModel.getDisplayFormat());
    emailField.setText("");
    switch (contactsModel.getMailFormat()) {
    case 0: htmlChoice.setSelected(true); break;
    case 1: plainTextChoice.setSelected(true); break;
    case 2: customChoice.setSelected(true); break;
    }
    javax.swing.DefaultListModel listModel = new javax.swing.DefaultListModel();
    Object[] mails = contactsModel.getEmails();
    for (int i=0; i<mails.length; i++) listModel.addElement(mails[i]);
    emailsList.setModel(listModel);
    details.pack();
    details.setVisible(true);
}
```

The IDE adds the listener to the Edit button enabling the button to interact with the contactsModel and database whenever clicked.



Setting the Event Handlers

To add an event handler for the Add button:

1. Right-click the Add button and choose Events > Action > actionPerformed.

2. In the Source Editor, select the line where the cursor is located and add the following code:

```
showDetailDialog(true);
```

To add an event handler for the Remove button:

1. Right-click the Remove button and choose Events > Action > actionPerformed.
2. In the Source Editor, select the line where the cursor is located and add the following code in the bod of the addContactActionPerformed method:

```
contactsModel.removeContact();
```

Setting the Table Selection Model

Now it's time to set the table selection model that will listen for propert changes in the model and determine if the selected contact can be edited or not. Note that for the purposes of the tutorial, the ContactsModel implementation permits an selected contact to be edited or removed.

To set the selection model for the table:

1. In the Inspector window, expand the contactTablePane node and select the JTable.
2. In the Properties window, click the ellipsis button for selectionModel propert (toward the bottom in the Other Properties section).
3. In the editor that appears, select the Propert radiobutton and click the ellipsis button.
4. In the Select Propert dialog that opens, choose contactsModel in the combobox. Then select contactSelection in the Properties pane.
5. Click OK to exit the dialogs.

To connect contactsModel with the Edit and Remove buttons:

1. Right-click the contactsModel node in the Inspector and choose Events > Propert Change > propert Change from the pop-up menu.
2. In the Source Editor, cop and paste the following code on the line where the cursor is positioned.

```
editContact.setEnabled(contactsModel.isEditingEnabled());
```

3. Repeat the procedure for the Remove button, but this time use the following code:

```
removeContact.setEnabled(contactsModel.isRemovalEnabled());
```

The IDE sets the buttons to enable editing and removal of the database contacts.

top

Creating the Application Logic

Though we've now managed to bind our database to our GUI using the provided JDBC components, we still need to set up the various events that will be fired when users click buttons or manipulate other GUI components. In this section, we'll focus on creating the application logic that will control how the various Swing components that make up our GUI will interact with the database.

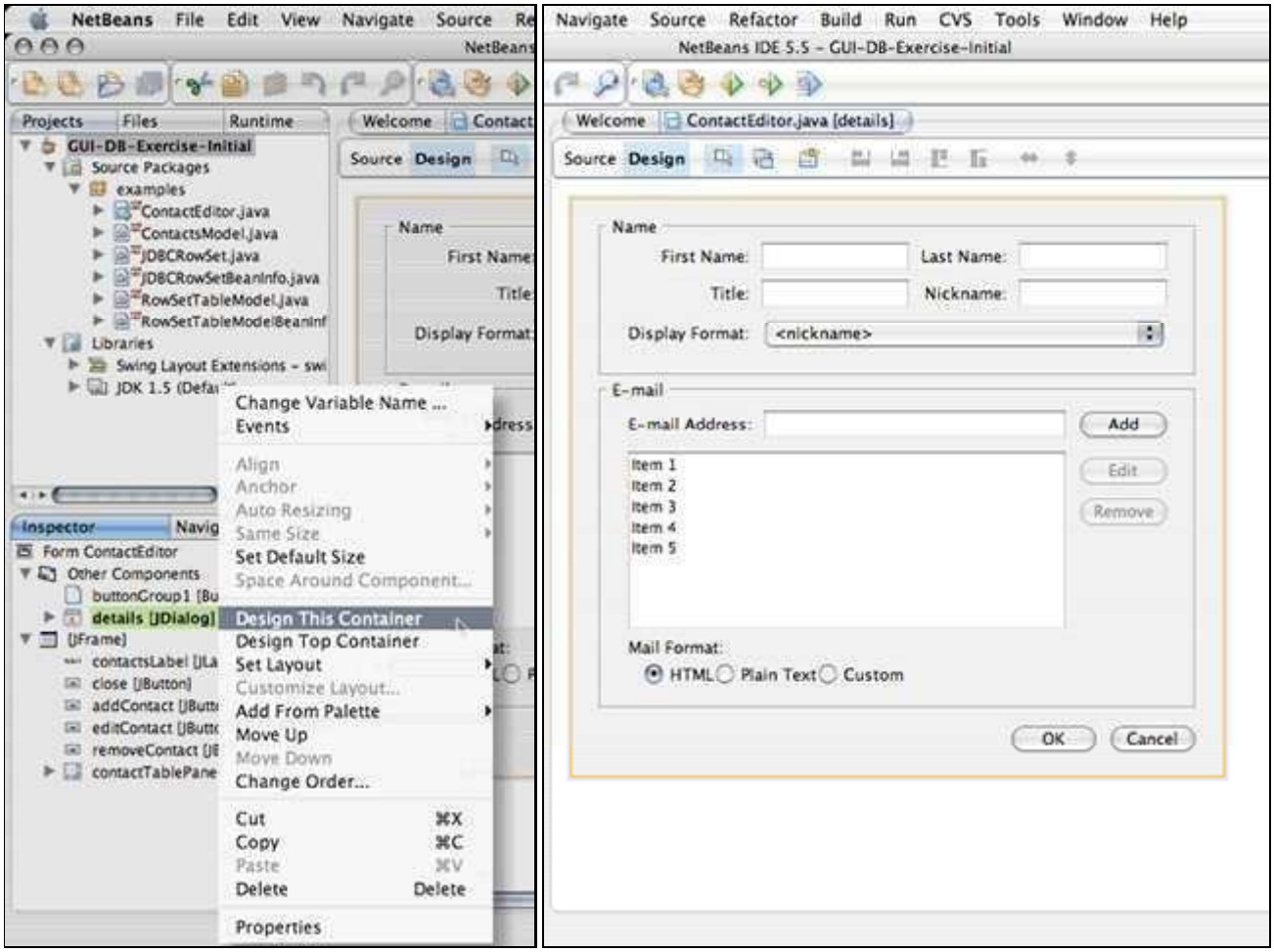
Changing GUI Builder Focus

Now that we've completed the necessar modifications to the top-level Contacts container that will be our application's main window, we need to shift the GUI Builder's focus to the details dialog in order to complete the process of binding the database to our GUI.

To change the GUI Builder's focus:

1. In the Inspector window, right-click the details dialog's node and choose Design This Container from the pop-up menu. Note that you can also simply double-click the node to accomplish this.

   The IDE shifts focus to the nested `details` JDialog container.



Changing GUI Builder Focus, Before and After

Setting Button Actions

Because buttons are our interface's primary means of interacting with the data contained in our database, we'll start by setting the events they will fire in the course of interacting with the database.

To edit event actions for the Add button:

1. Right-click the Add button and select Events > Action > actionPerformed.
2. In the Source Editor, paste the following code at the insertion point:

```
((javax.swing.DefaultListModel)emailsList.getModel()).addElement(emailField.getText());
```

   The IDE sets the Add button to get the list of email addresses when it is clicked.

To edit event actions for the Edit button:

1. Right-click the Edit button and select Events > Action > actionPerformed.
2. In the Source Editor, paste the following code at the insertion point:

```
((javax.swing.DefaultListModel)emailsList.getModel()).setElementAt(emailField.getText(),
emailsList.getSelectedIndex());
```

The IDE sets the Edit button to set the list of email addresses whenever clicked.

To edit event actions for the Remove button:

1. Right-click the Remove button and select Events > Action > actionPerformed.
2. In the Source Editor, paste the following code at the insertion point.

```
((javax.swing.DefaultListModel)emailsList.getModel()).removeElementAt(emailsList.getSelectedIndex());
```

The IDE sets the Remove button to remove the selected email address when clicked.

Defining the JList Event Listener

Now we need to add an event listener that will watch for changes to the database. Though the JList is located within the Details dialog, we'll set the component's event listener using the Inspector window. This approach can be particularly useful when setting listeners for components existing in a different part of the form which would otherwise require switching the GUI Builder's focus.

To set the JList's event listener:

1. Right-click the `emailsList` JList (which is nested within the listScrollPane component) and select Events > ListSelection > valueChanged.
2. In the Source Editor, paste the following code at the insertion point:

```
editEmail.setEnabled(emailsList.getSelectedIndex() != -1);
removeEmail.setEnabled(emailsList.getSelectedIndex() != -1);
emailField.setText((String)emailsList.getSelectedValue());
```

The IDE sets the JList's event listener to watch for changes to the database data.

Editing Event Actions (Revisited)

In this section we'll set up the final two buttons our Contact Details GUI requires and take a look at using the GUI Builder's Connection Wizard to set event actions.

To edit event actions for the OK button:

1. Right-click the OK button and select Events > Action > actionPerformed.
2. In the Source Editor, paste the following code at the insertion point:

```
contactsModel.updateContact(firstNameField.getText(), lastNameField.getText(),
titleField.getText(), nicknameField.getText(), displayFormat.getSelectedIndex(),
htmlChoice.isSelected() ? 0 : (plainTextChoice.isSelected() ? 1 : 2),
((javax.swing.DefaultListModel)emailsList.getModel()).toArray());
details.setVisible(false);
```

So far we've been relying on copying and pasting to set up our program's application logic, but the IDE also includes a tool designed to assist in this common task. This time we'll use the GUI Builder's Connection Wizard to set the Contact Details Cancel button to close the dialog and discard any changes that have been made.

To edit event actions for the Cancel button:

1. Click the Connection Wizard button in the GUI Builder's toolbar.
2. Select the Cancel button and then the details dialog.
3. In the Connection wizard that appears, confirm that `cancelButton` is specified as the Source Component.
4. Select action > actionPerformed in the Select Source Event pane and click Next.
5. On the Specify Target Operation page of the wizard, confirm that `details` is specified as the Target Component. Then select the Method Call radiobutton and choose `setVisible(boolean)` from the list of properties. Click Next.
6. On the Enter Parameters wizard page, select the Value radiobutton and enter `false`. Click Finish.
   The IDE's Connection Wizard sets the dialog's Cancel button to exit the dialog whenever clicked.

top

Finishing Up

Now that we've managed to rough out our ContactEditor application's GUI, we'll take a look at the results of our hand work.

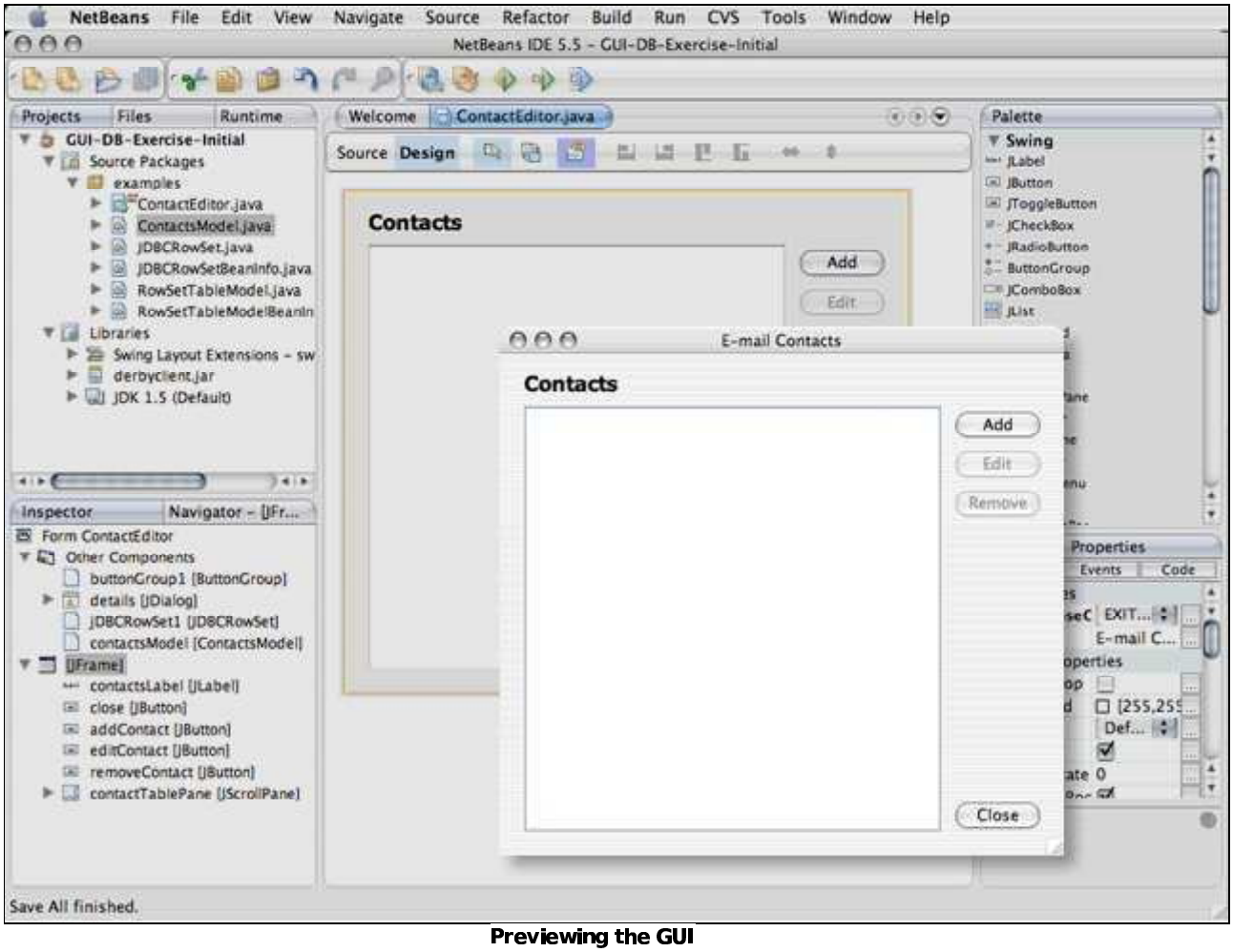
Previewing the GUI

Now that we've integrated the details dialog with the main Contacts Application window and connected them to our Contacts database, we can preview the UI to see if ever thing looks right prior to building and running.

To preview the GUI:

1. Click the Preview button in the GUI Builder's toolbar.

   The IDE opens the GUI in its own window, enabling ou to test the form's appearance and resizing behavior. Notice that even though we left the details dialog as the container that had focus in the IDE's Design Area, the application's main window appears when clicking the Preview button.



Previewing the GUI

You ma have noticed over the course of the tutorial that though the main window JFrame's title propert is set to Contacts, no title has been visible in the GUI Builder's Design Area. This is because titlebars are rendered d namicall at runtime so ou'll need to click the Preview Form button if ou want to see the results of title propert edits.

Testing Database Functionality

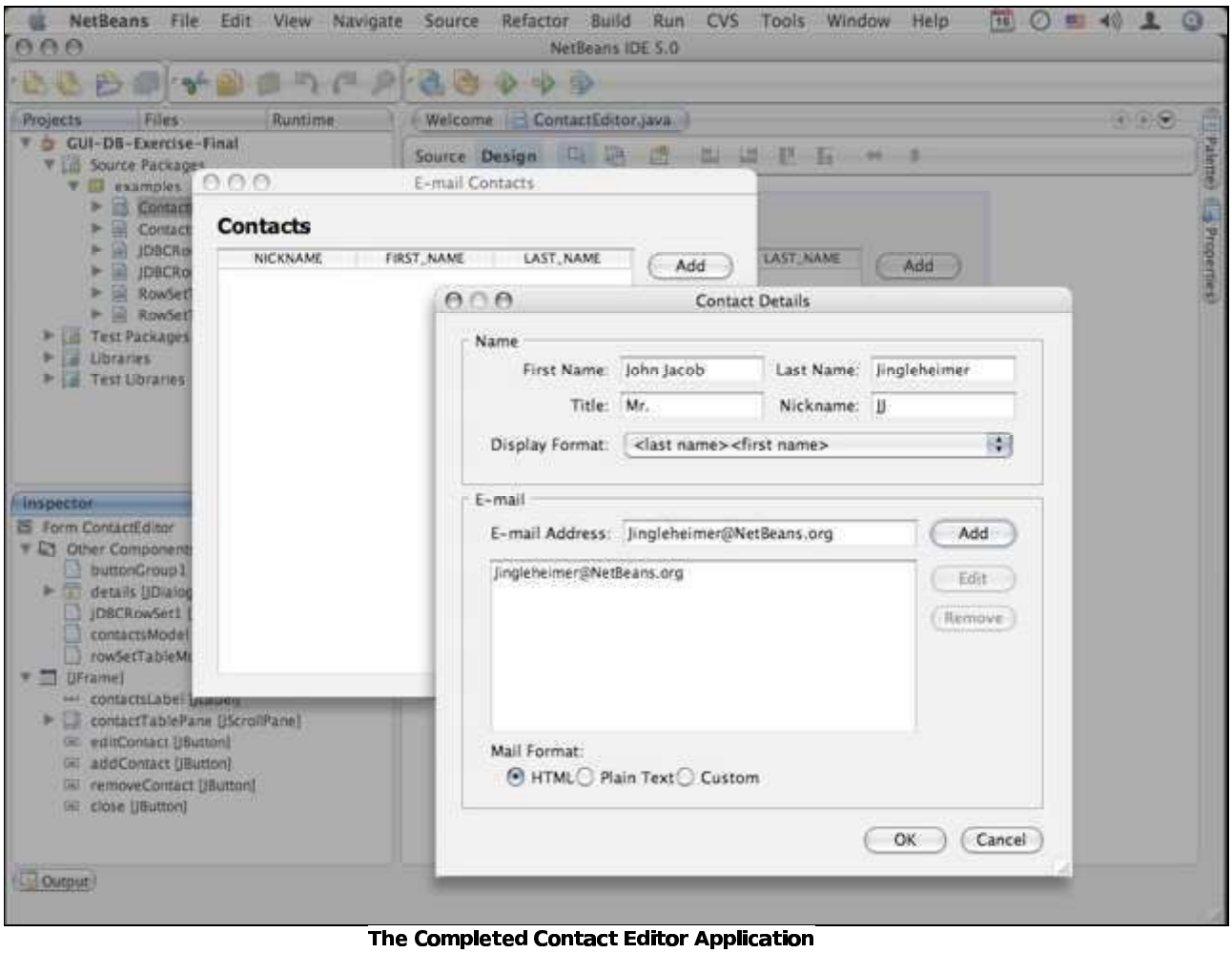
Now that we've verified that our GUI looks like it should, we'll test our application to verif that it works as we expect it to. While the preview feature is helpful for identif ing issues with visual aspects of our GUI (alignments, resizeabilit behavior, and so forth), we have to build and run the application if we want to view an nested windows included in our GUI or test the business logic.

To build and run the application:

1. Right-click the project node in the Projects window and choose Run Project. Alternatel , ou can click the Run Main Project button in the GUI Builder's toolbar.

   The IDE builds the application, reporting the status and an problems encountered in the Output window. Once the build succeeds, the IDE launches the application and the GUI's main window appears.

Now that the application is running, ou can further test the GUI as well as put its various interactions with the database itself through the paces b adding the names of our friends and famil as shown in the following image.



The Completed Contact Editor Application

top

Next Steps

You have now completed the IDE's GUI Building tutorial. For more information on working with Java GUI's in the NetBeans IDE, see:

- Working with the Java DB (Derb) Database in NetBeans 5.5**.** A tutorial about setting up and using the Java DB Database in NetBeans IDE.
- GUI Building in the NetBeans IDE**.** A quick guide through the process of creating graphical user interfaces with the NetBeans IDE.
- Matisse GUI Builder FAQ**.** A collection of useful tips and tricks for using NetBeans GUI Builder.
- NetBeans Support and Docs**.** Full list of articles, FAQs, and mailing lists for NetBeans IDE users.

top