**1 Why Use FXML**

FXML is an XML-based language that provides the structure for building a user interface separate from the application logic of your code. This separation of the presentation and application logic is attractive to web developers because they can assemble a user interface that leverages Java components without mastering the code for fetching and filling in the data.

The following sections provide more information about FXML, and when you would choose FXML over other methods of creating a user interface.

* [Introduction to FXML](http://docs.oracle.com/javafx/2/fxml_get_started/why_use_fxml.htm#CIHJCABJ)
* [Simple Example of FXML](http://docs.oracle.com/javafx/2/fxml_get_started/why_use_fxml.htm#CHDGAFHF)
* [Benefits of FXML](http://docs.oracle.com/javafx/2/fxml_get_started/why_use_fxml.htm#CHDIEGBB)
* [FXML and Scene Builder](http://docs.oracle.com/javafx/2/fxml_get_started/why_use_fxml.htm#BABIECHG)

**Introduction to FXML**

FXML does not have a schema, but it does have a basic predefined structure. What you can express in FXML, and how it applies to constructing a scene graph, depends on the API of the objects you are constructing. Because FXML maps directly to Java, you can use the API documentation to understand what elements and attributes are allowed. In general, most JavaFX classes can be used as elements, and most Bean properties can be used as attributes.

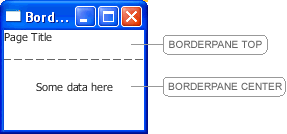
From a Model View Controller (MVC) perspective, the FXML file that contains the description of the user interface is the view. The controller is a Java class, optionally implementing the Initializable class, which is declared as the controller for the FXML file. The model consists of domain objects, defined on the Java side, that you connect to the view through the controller. An example of this structure is in the tutorial [Creating an Address Book with FXML](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACFEHBI).

While you can use FXML to create any user interface, FXML is particularly useful for user interfaces that have large, complex scene graphs, forms, data entry, or complex animation. FXML is also well suited for defining static layouts such as forms, controls, and tables. In addition, you can use FXML to construct dynamic layouts by including scripts.

**Simple Example of FXML**

The easiest way to show the advantages of FXML is with an example. Take a look at [Figure 1-1](http://docs.oracle.com/javafx/2/fxml_get_started/why_use_fxml.htm#CHDJCIFB), which shows a user interface that includes a border pane layout that has a top and center region, each of which contains a label.

***Figure 1-1 Border Pane Simple Example***

  
[Description of "Figure 1-1 Border Pane Simple Example"](http://docs.oracle.com/javafx/2/fxml_get_started/img_text/borderpane_simple_example.htm)

First, look at how the user interface is constructed and built directly in the source code, as shown in [Example 1-1](http://docs.oracle.com/javafx/2/fxml_get_started/why_use_fxml.htm#CHDGCFJJ).

***Example 1-1 Java Code for a User Interface***

BorderPane border = new BorderPane();

Label toppanetext = new Label("Page Title");

border.setTop(toppanetext);

Label centerpanetext = new Label ("Some data here");

border.setCenter(centerpanetext);

Now look at [Example 1-2](http://docs.oracle.com/javafx/2/fxml_get_started/why_use_fxml.htm#CHDBCBEG), which shows the same user interface, but in FXML markup. Here you can see the hierarchical structure of the user interface, which in turn makes it easier to add components and build upon the user interface.

***Example 1-2 FXML Markup for a User Interface***

<BorderPane>

<top>

<Label text="Page Title"/>

</top>

<center>

<Label text="Some data here"/>

</center>

</BorderPane>

**Benefits of FXML**

In addition to providing web developers a familiar approach to designing user interfaces, FXML offers these benefits:

* Because the scene graph is more transparent in FXML, it is easy for a development team to create and maintain a testable user interface.
* FXML is not a compiled language; you do not need to recompile the code to see the changes.
* The content of an FXML file can be localized as the file is read. For example, if an FXML file is loaded using the en\_US locale, it produces the string "First Name" for a label based on the following resource string:

<Label text="%firstName"/>

If the locale is changed to fr\_FR and the FXML file is reloaded, the label shows "Prénom".

The same is not true for Java code, because you must manually update the content of every element of your user interface by obtaining a reference to it and calling the appropriate setter (such as setText()).

* You can use FXML with any Java Virtual Machine (JVM) language, such as Java, Scala, or Clojure.
* FXML is not limited to the view portion of the MVC model. You can construct services or tasks or domain objects, and you can use JavaScript or other scripting languages in FXML. For an example of using JavaScript, see [Use a Scripting Language to Handle Events](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#JFXGS213) in the FXML tutorial of the Getting Started guide.

**FXML and Scene Builder**

Just as some developers prefer to work directly in the XML code, other developers prefer to use a tool to author their XML. The same is true with FXML.

If you prefer to use a tool, or if you want to create a quick prototype for purposes such as getting feedback, then you might prefer to use JavaFX Scene Builder. Scene Builder is a design tool that generates the FXML source code as you define the user interface for your application. Scene Builder can help you to quickly prototype an interactive application that connect components to the application logic. For more information, see [Getting Started with JavaFX Scene Builder](http://docs.oracle.com/javafx/scenebuilder/1/get_started/jsbpub-get_started.htm).

**3 Creating an Address Book with FXML**

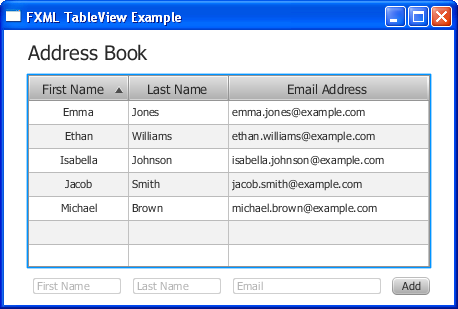
In this tutorial, you create an Address Book application that includes a table of names and email addresses, as shown in [Figure 3-1](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACFCJGI). The tutorial shows how to populate a table with data, sort the data at application startup, align the data in the table cells, and add rows to the table.

Some amount of knowledge of FXML and application development is assumed for this tutorial. Before you start, you should have completed the [FXML tutorial in the Getting Started series](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm), because it teaches the basics of FXML development. Specifically, for the Address Book tutorial, you should know:

* The basic structure of an FXML project (.java, .fxml, and controller files)
* How to create and run a JavaFX FXML project in NetBeans IDE
* The basics of layout and user interface components

Before you begin this tutorial, ensure that the version of NetBeans IDE that you are using supports your version of JavaFX 2. See the [System Requirements](http://docs.oracle.com/javafx/release-documentation.html) for details.

***Figure 3-1 Address Book Application***

  
[Description of "Figure 3-1 Address Book Application"](http://docs.oracle.com/javafx/2/fxml_get_started/img_text/table_final.htm)

**Set Up the Project**

Your first task is to set up a JavaFX FXML project in NetBeans IDE.

1. From the **File** menu, choose **New Project**.
2. In the **JavaFX** application category, choose **JavaFX FXML Application**. Click **Next**.
3. Name the project **FXMLTableView** and click **Finish**.

NetBeans IDE opens an FXML project that includes the code for a basic Hello World application. The application includes three files: FXMLTableView.java, Sample.fxml, and Sample.java.

1. Rename Sample.java to FXMLTableViewController.java so that the name is more meaningful for this application.
   1. In the Projects window, right-click **Sample.java** and choose **Refactor** then **Rename**.
   2. Enter **FXMLTableViewController**, and then click **Refactor**.
2. Rename Sample.fxml to fxml\_tableview.fxml.
   1. Right-click **Sample.fxml** and choose **Rename**.
   2. Enter **fxml\_tableview** and click **OK**.
3. Open FXMLTableView.java and edit the FXMLTableView class to look like [Example 3-1](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACHBAEJ).

***Example 3-1 FXMLTableView.java***

public class FXMLTableView extends Application {

@Override

public void start(Stage primaryStage) throws Exception {

primaryStage.setTitle("FXML TableView Example");

primaryStage.setScene

((Scene)FXMLLoader.load(getClass().getResource("fxml\_tableview.fxml")));

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

Note that the code for the scene has been removed from the Java file. In the next section of the tutorial, [Create the Basic User Interface](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACBGIIG), you will add the code for the scene in the FXML file.

1. Press Ctrl (or Cmd) + Shift + I to fix the import statements.

**Create the Basic User Interface**

Now, define the user interface by creating the scene and adding a GridPane layout container as the root node. Then add a Label and a TableView component as child nodes of the grid pane.

1. Open the fxml\_tableview.fxml file.
2. Delete the <AnchorPane> markup that NetBeans IDE auto-generated.
3. Add code for the scene as shown in [Example 3-2](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACCACJI).

***Example 3-2 Scene***

<Scene width="550" height="550"

fx:controller="fxmltableview.FXMLTableViewController"

xmlns:fx="http://javafx.com/fxml">

</Scene>

You can ignore the error "File not found in the specified address: http://javafx.com/fxml" that might appear in the output window.

1. Add a GridPane layout container as the root node of the scene, using the code in [Example 3-3](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACCCGFD).

***Example 3-3 GridPane***

<Scene width="550" height="550"

fx:controller="fxmltableview.FXMLTableViewController"

xmlns:fx="http://javafx.com/fxml">

**<GridPane alignment="center" hgap="10" vgap="10">**

**<padding>**

**<Insets top="10" right="10" bottom="10" left="10"/>**

**</padding>**

**</GridPane>**

</Scene>

1. Add a Label and TableView component to the grid pane. The code is in [Example 3-4](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACECECI).

***Example 3-4 Label and TableView***

<GridPane alignment="center" hgap="10" vgap="10">

<padding>

<Insets top="10" right="10" bottom="10" left="10"/>

</padding>

**<Label text="Address Book" GridPane.columnIndex="0" GridPane.rowIndex="0"**

**style="-fx-font: NORMAL 20 Tahoma;"/>**

**<TableView fx:id="tableView" GridPane.columnIndex="0"**

**GridPane.rowIndex="1">**

**</TableView>**

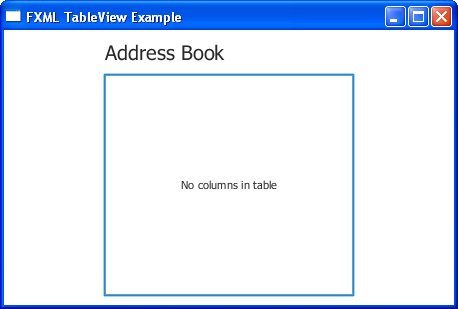
</GridPane>

1. Add the import statement for the Insets class.

<?import javafx.geometry.Insets?>

1. Run the program. You see the label Address Book and a table with the text "No columns in table," which is the default caption defined by the TableView implementation, as shown in [Figure 3-2](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACBDJCD).

***Figure 3-2 Table with No Columns***

  
[Description of "Figure 3-2 Table with No Columns"](http://docs.oracle.com/javafx/2/fxml_get_started/img_text/table_no_columns.htm)

**Add Columns to the Table**

Now, use the TableColumn class to add three columns for displaying the data: First Name, Last Name, and Email address. The code is in [Example 3-5](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACIAAFE).

***Example 3-5 Table Columns***

<TableView fx:id="tableView" GridPane.columnIndex="0" GridPane.rowIndex="1">

**<columns>**

**<TableColumn text="First Name">**

**</TableColumn>**

**<TableColumn text="Last Name">**

**</TableColumn>**

**<TableColumn text="Email Address">**

**</TableColumn>**

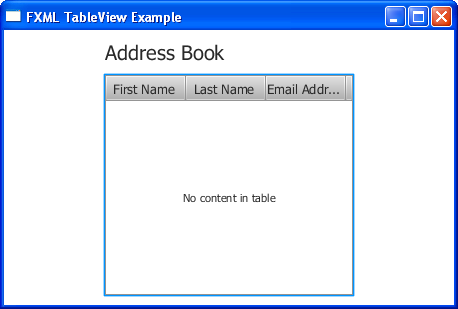
**</columns>**

</TableView>

**Tip**: For more information on the TableColumn class or any other JavaFX class discussed in this tutorial, refer to the [API documentation](http://docs.oracle.com/javafx/2/api/index.html).

[Figure 3-3](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACGDIJJ) shows the table with the columns for First Name, Last Name, and Email Address.

***Figure 3-3 Address Book with Three Columns***

  
[Description of "Figure 3-3 Address Book with Three Columns"](http://docs.oracle.com/javafx/2/fxml_get_started/img_text/table_columns.htm)

**Define the Data Model**

When you create a table in a JavaFX application, it is a best practice to implement a class that defines the data model and provides methods and fields to further work with the table. Here, you create a Person class to define the data for the address book.

1. In NetBeans IDE, right-click the **fxmltableview** folder under Source Packages, and choose **New** then **Java Class**.
2. Name the class **Person** and then click **Finish**.
3. Implement a Person class to define the data, as shown in [Example 3-6](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACHHCAD).

***Example 3-6 Person Class***

package fxmltableview;

import javafx.beans.property.SimpleStringProperty;

public class Person {

private final SimpleStringProperty firstName = new SimpleStringProperty("");

private final SimpleStringProperty lastName = new SimpleStringProperty("");

private final SimpleStringProperty email = new SimpleStringProperty("");

public Person() {

this("", "", "");

}

public Person(String firstName, String lastName, String email) {

setFirstName(firstName);

setLastName(lastName);

setEmail(email);

}

public String getFirstName() {

return firstName.get();

}

public void setFirstName(String fName) {

firstName.set(fName);

}

public String getLastName() {

return lastName.get();

}

public void setLastName(String fName) {

lastName.set(fName);

}

public String getPrimary() {

return getEmail();

}

public String getSecondary() {

return getEmail();

}

public String getEmail() {

return email.get();

}

public void setEmail(String fName) {

email.set(fName);

}

}

**Associate Data with the Table Columns**

The next tasks are to define rows for the data and associate the data with the table columns. You add this code in the FXML file.

1. In the fxml\_tableview.fxml file, create an ObservableList array and define as many data rows as you would like to show in your table. Sample code is in [Example 3-7](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACFHDEJ). Add the code between the </columns> and </TableView> markup.

***Example 3-7 ObservableList Array***

</columns>

**<items>**

**<FXCollections fx:factory="observableArrayList">**

**<Person firstName="Jacob" lastName="Smith"**

**email="jacob.smith@example.com"/>**

**<Person firstName="Isabella" lastName="Johnson"**

**email="isabella.johnson@example.com"/>**

**<Person firstName="Ethan" lastName="Williams"**

**email="ethan.williams@example.com"/>**

**<Person firstName="Emma" lastName="Jones"**

**email="emma.jones@example.com"/>**

**<Person firstName="Michael" lastName="Brown"**

**email="michael.brown@example.com"/>**

**</FXCollections>**

**</items>**

</TableView>

1. Specify a cell factory for each column to associate the data with the column, as shown in [Example 3-8](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACHDCCF).

***Example 3-8 Cell Factories***

<columns>

<TableColumn text="First Name">

**<cellValueFactory><PropertyValueFactory property="firstName" />**

**</cellValueFactory>**

</TableColumn>

<TableColumn text="Last Name">

**<cellValueFactory><PropertyValueFactory property="lastName" />**

**</cellValueFactory>**

</TableColumn>

<TableColumn text="Email Address">

**<cellValueFactory><PropertyValueFactory property="email" />**

**</cellValueFactory>**

</TableColumn>

</columns>

Cell factories are implemented by using the PropertyValueFactory class, which uses the firstName, lastName, and email properties of the table columns as references to the corresponding methods of the Person class.

1. Import the required packages:

***Example 3-9 Import Statements***

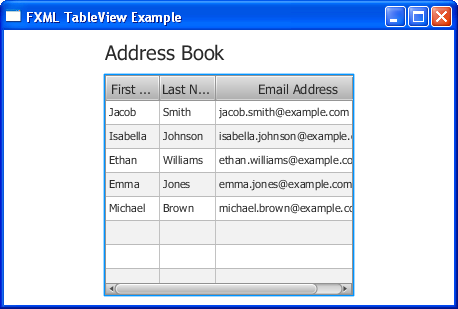
<?import javafx.scene.control.cell.\*?>

<?import javafx.collections.\*?>

<?import fxmltableview.\*?>

Running the application at this point shows the table populated with data, as shown in [Figure 3-4](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACFJBJH).

***Figure 3-4 Table with Data***

  
[Description of "Figure 3-4 Table with Data"](http://docs.oracle.com/javafx/2/fxml_get_started/img_text/table_with_data.htm)

Here are some built-in features of the TableView class for you to try:

* Resize a column width by dragging the column divider in the table header to the left or right.
* Move a column by dragging the column header.
* Alter the sort order of data by clicking in a column header. The first click enables ascending sorting order, the second click enables descending sorting order, and the third click disables sorting. By default, no sorting is applied.

**Set Sort Order on Startup**

In this task, you set the sort order so that the entries in the First Name column appear in ascending alphabetical order on application startup. You do this by creating an ID for the table column and then setting up a reference to it.

1. Add an id to the First Name column:

<TableColumn fx:id="firstNameColumn" text="First Name">

1. Specify the sort order by adding the code in [Example 3-10](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACFIIFE) between the </items> and </TableView> markup.

***Example 3-10 Sort Order***

</items>

**<sortOrder>**

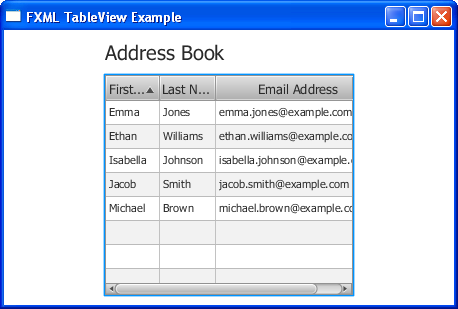
**<fx:reference source="firstNameColumn"/>**

**</sortOrder>**

</TableView>

You can see the results in [Figure 3-5](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACEDHHI).

***Figure 3-5 Table with First Column Data Sorted at Startup***

  
[Description of "Figure 3-5 Table with First Column Data Sorted at Startup"](http://docs.oracle.com/javafx/2/fxml_get_started/img_text/table_sort.htm)

**Define Column Widths**

Here, you add the prefWidth property to increase the column widths, as shown in [Example 3-11](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACFIABH).

***Example 3-11 Column Widths***

<TableColumn fx:id="firstnameColumn" text="First Name" **prefWidth="100"**>

<cellValueFactory><PropertyValueFactory property="firstName" />

</cellValueFactory>

</TableColumn>

<TableColumn text="Last Name" **prefWidth="100"**>

<cellValueFactory><PropertyValueFactory property="lastName" />

</cellValueFactory>

</TableColumn>

<TableColumn text="Email Address" **prefWidth="200"**>

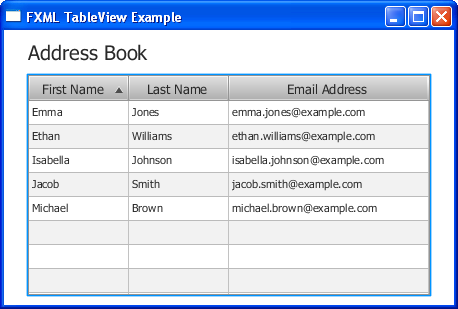
<cellValueFactory><PropertyValueFactory property="email" />

</cellValueFactory>

</TableColumn>

The result is in [Figure 3-6](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACEDCFD). The column widths have been increased so that all data is visible in each table row.

***Figure 3-6 Table with Column Widths Set***

  
[Description of "Figure 3-6 Table with Column Widths Set"](http://docs.oracle.com/javafx/2/fxml_get_started/img_text/table_prefwidth.htm)

**Set Alignment in Table Cells**

Another customization you can do is to set the alignment of the data in the table cells. You implement the logic in a new class named FormattedTableCellFactory and then set the alignment in the <TableColumn> markup in the FXML code.

1. In NetBeans IDE, right-click the **fxmltableview** folder under Source Packages, and choose **New** then **Java Class**.
2. Name the class **FormattedTableCellFactory** and then click **Finish**.
3. Modify the FormattedTableCellFactory class by implementing the Callback class and creating instances of the TextAlignment and Format classes, as shown in [Example 3-12](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACCJICD). The S parameter is the TableView generic type and the T parameter is the type of the content of all cells in this table column.

***Example 3-12 Callback Class***

public class FormattedTableCellFactory<S, T>

implements Callback<TableColumn<S, T>, TableCell<S, T>> {

private TextAlignment alignment;

private Format format;

public TextAlignment getAlignment() {

return alignment;

}

public void setAlignment(TextAlignment alignment) {

this.alignment = alignment;

}

public Format getFormat() {

return format;

}

public void setFormat(Format format) {

this.format = format;

}

1. Implement the TableCell and TableColumn classes by appending the code in [Example 3-13](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACIEEHH). This code overrides the updateItem method of the TableCell class and calls the setTextAlignment method on the table cell.

***Example 3-13 TableCell and TableColumn Classes***

@Override

@SuppressWarnings("unchecked")

public TableCell<S, T> call(TableColumn<S, T> p) {

TableCell<S, T> cell = new TableCell<S, T>() {

@Override

public void updateItem(Object item, boolean empty) {

if (item == getItem()) {

return;

}

super.updateItem((T) item, empty);

if (item == null) {

super.setText(null);

super.setGraphic(null);

} else if (format != null) {

super.setText(format.format(item));

} else if (item instanceof Node) {

super.setText(null);

super.setGraphic((Node) item);

} else {

super.setText(item.toString());

super.setGraphic(null);

}

}

};

cell.setTextAlignment(alignment);

switch (alignment) {

case CENTER:

cell.setAlignment(Pos.CENTER);

break;

case RIGHT:

cell.setAlignment(Pos.CENTER\_RIGHT);

break;

default:

cell.setAlignment(Pos.CENTER\_LEFT);

break;

}

return cell;

}

}

1. Fist the import statements.
2. In the fxml\_tableview.fxml file, create a cellFactory alignment for the First Name column by adding the following code under the <cellValueFactory> markup.

***Example 3-14 Alignment in Data Cell***

<TableColumn fx:id="firstNameColumn" text="First Name" prefWidth="100">

<cellValueFactory><PropertyValueFactory property="firstName" />

</cellValueFactory>

**<cellFactory>**

**<FormattedTableCellFactory alignment="center">**

**</FormattedTableCellFactory>**

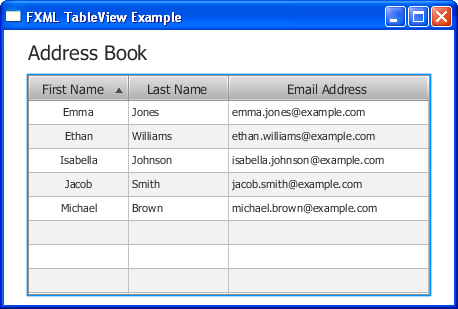
**</cellFactory>**

</TableColumn>

You can create cellFactory alignment for the remaining columns using left, right, or center.

Running the project now shows data center-aligned in the First Name column.

***Figure 3-7 Data Center Aligned in First Name Column***

  
[Description of "Figure 3-7 Data Center Aligned in First Name Column"](http://docs.oracle.com/javafx/2/fxml_get_started/img_text/table_alignment.htm)

**Add Rows to the Table**

Now, you add the ability for users to add a row of data to the table. You add the application logic in the Controller class. Then, you modify the user interface to include three text fields and a button for entering the data.

1. Open the FXMLTableViewController.java file.
2. Edit the FXMLTableViewController class so it looks like the code in [Example 3-15](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACHFIJJ).

***Example 3-15 FXMLTableViewController.java***

public class FXMLTableViewController {

@FXML private TableView<Person> tableView;

@FXML private TextField firstNameField;

@FXML private TextField lastNameField;

@FXML private TextField emailField;

@FXML

protected void addPerson(ActionEvent event) {

ObservableList<Person> data = tableView.getItems();

data.add(new Person(firstNameField.getText(),

lastNameField.getText(),

emailField.getText()

));

firstNameField.setText("");

lastNameField.setText("");

emailField.setText("");

}

}

1. Fix the import statements. They should be:

***Example 3-16 Import Statements in FXMLTableViewController***

import javafx.collections.ObservableList;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.scene.control.TableView;

import javafx.scene.control.TextField;

1. In the fxml\_tableview.fxml file, add this code before the </GridPane> markup

***Example 3-17 Text Fields and Button for Adding a Row***

</TableView>

**<HBox spacing="10" alignment="bottom\_right" GridPane.columnIndex="0"**

**GridPane.rowIndex="2">**

**<TextField fx:id="firstNameField" promptText="First Name"**

**prefWidth="90"/>**

**<TextField fx:id="lastNameField" promptText="Last Name"**

**prefWidth="90"/>**

**<TextField fx:id="emailField" promptText="email"**

**prefWidth="150"/>**

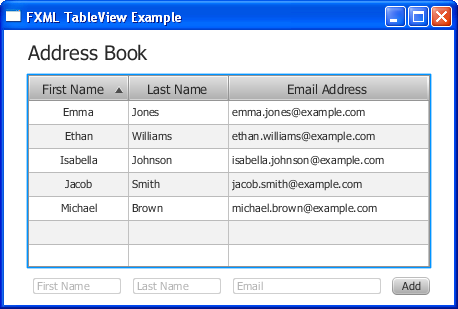
**<Button text="Add" onAction="#addPerson"/>**

**</HBox>**

</GridPane>

Run the application and you see that the text fields and button have been appear below the table, as shown in [Figure 3-8](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm#CACCEFJI). Enter data in the text fields and click **Add** to see the application in action.

***Figure 3-8 Table with Text Fields and Button for Adding Data***

  
[Description of "Figure 3-8 Table with Text Fields and Button for Adding Data"](http://docs.oracle.com/javafx/2/fxml_get_started/img_text/table_add_row_.htm)

**Where to Go from Here**

This concludes the FXML TableView tutorial, but here are some things for you to try next.

* Provide a filter to check if data was entered in the correct format.
* Customize the table by applying a cascading style sheet to distinguish between empty and non-empty rows. Refer to "Styling UI Controls with CSS" in [JavaFX UI Controls](http://docs.oracle.com/javafx/2/ui_controls/overview.htm) for more information.
* Enable editing of data in the table. See [Editing Data in the Table](http://docs.oracle.com/javafx/2/ui_controls/table-view.htm) in Using JavaFX UI Controls for pointers.
* See [Deployment of FXML Applications](http://docs.oracle.com/javafx/2/fxml_get_started/fxml_deployment.htm#BABJBHJJ) for additional deployment options.
* Look at [Introduction to FXML](http://docs.oracle.com/javafx/2/api/javafx/fxml/doc-files/introduction_to_fxml.html), which provides more information on the elements that make up the FXML language. The document is included in the javafx.fxml package in the API documentation at <http://docs.oracle.com/javafx/2/api/javafx/fxml/doc-files/introduction_to_fxml.html>
* For an example of an FXML application that draws data from a database, take a look at the Henley Sales Application sample by downloading the JavaFX Samples zip file at  
  <http://www.oracle.com/technetwork/java/javafx/downloads/>

The NetBeans projects and source code for this sample (called DataApp) are included in the samples zip file. See the readme for instructions on how to set up and run the application.

This sample provides several examples of how to populate a table from a database. In particular, look at the following files:

* + DataAppClient\src\com\javafx\experiments\dataapp\client\historytab\history-tab.fxml
  + DataAppClient\src\com\javafx\experiments\dataapp\client\livetab\live-tab.fxml
  + DataAppClient\src\com\javafx\experiments\dataapp\client\productstab\products-tab.fxml