NetBeans JavaFX Composer in Action
Creating Applications with Rich UI

David Kaspar
Software Engineer

JavaOne|Oracle Develop
NetBeans JavaFX Composer in Action
Creating Applications with Rich UI
Hands-on Lab: S313803

Introduction

NetBeans JavaFX Composer allows developers to create JavaFX applications visually by composing existing UI controls, layouts and effects. The integrated Data Source library offers connectivity to data from Web Services, databases and other sources. A state-based UI concept enriched with transitions provides easy development of a dynamic UI.

In this hands-on lab we are going to develop several real-world applications using the tool while showing some of the UI concepts and best practices for effective design of rich applications. We will discover how to do the following:

- Design the dynamic UI
- Get data from various data sources

Prerequisites

This hands-on lab assumes you have basic experience with JavaFX technology and NetBeans IDE.

Lab Exercises

- Exercise 1: Create and Run Project
- Exercise 2: Application States – Master
- Exercise 3: Application States – State
- Exercise 4: Data Source
- Exercise 5: Data Source Customized
- Exercise 6: Multiple State Variables

The directory where the lab content is placed contains:

- the “exercises” folder contains folders with NetBeans projects which you can use as a starting point for a particular exercise,
- the “solutions” folder contains folders with NetBeans projects which can be taken as a complete solution for a particular exercise,
- the “snippets.txt” file contains all code snippets that are listed in this document.

Please feel free to seek assistance from the instructor or Oracle Demo staff at any point during the lab.

Exercise 1: Create and Run Project

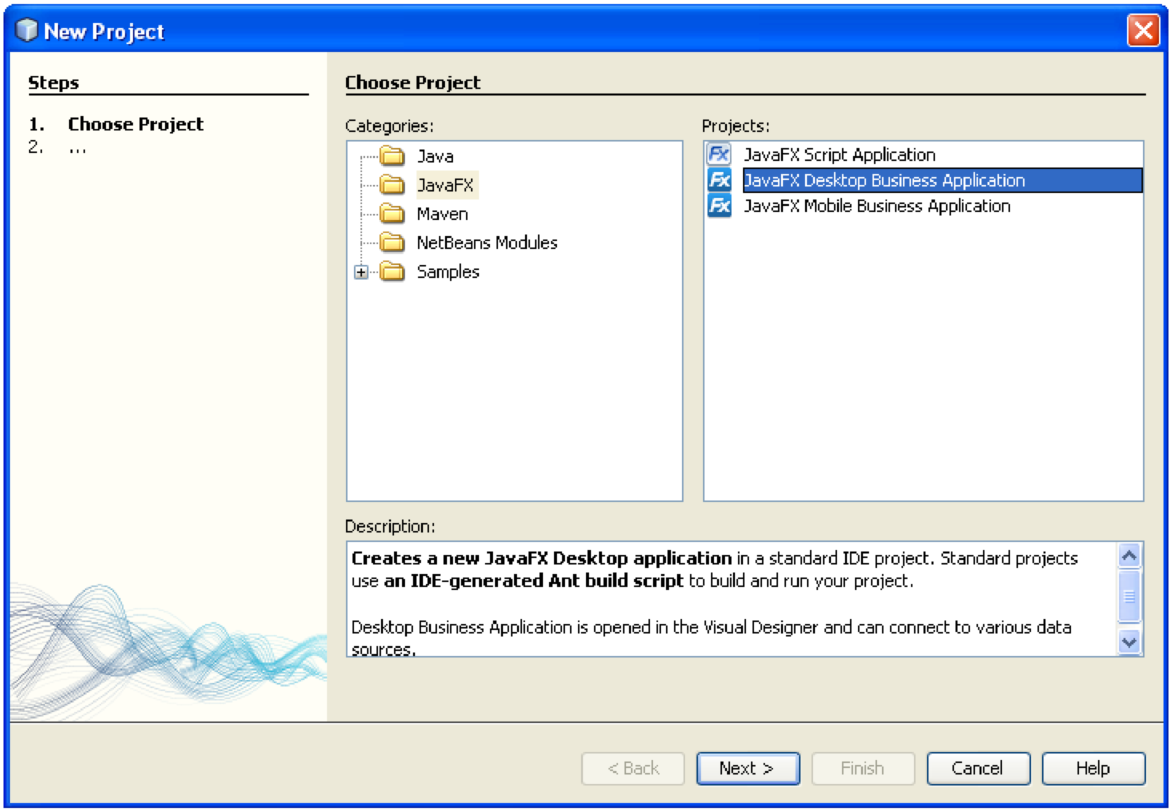
In this exercise, we are going to create a simple "Hello World" application that provides a basic overview of NetBeans JavaFX support and the NetBeans JavaFX Composer tool.

Create JavaFX Project

Invoke the "File | New Project..." main menu action. The New Project wizard appears.

Select "JavaFX" in "Categories". The following project types appear In the "Projects" list:

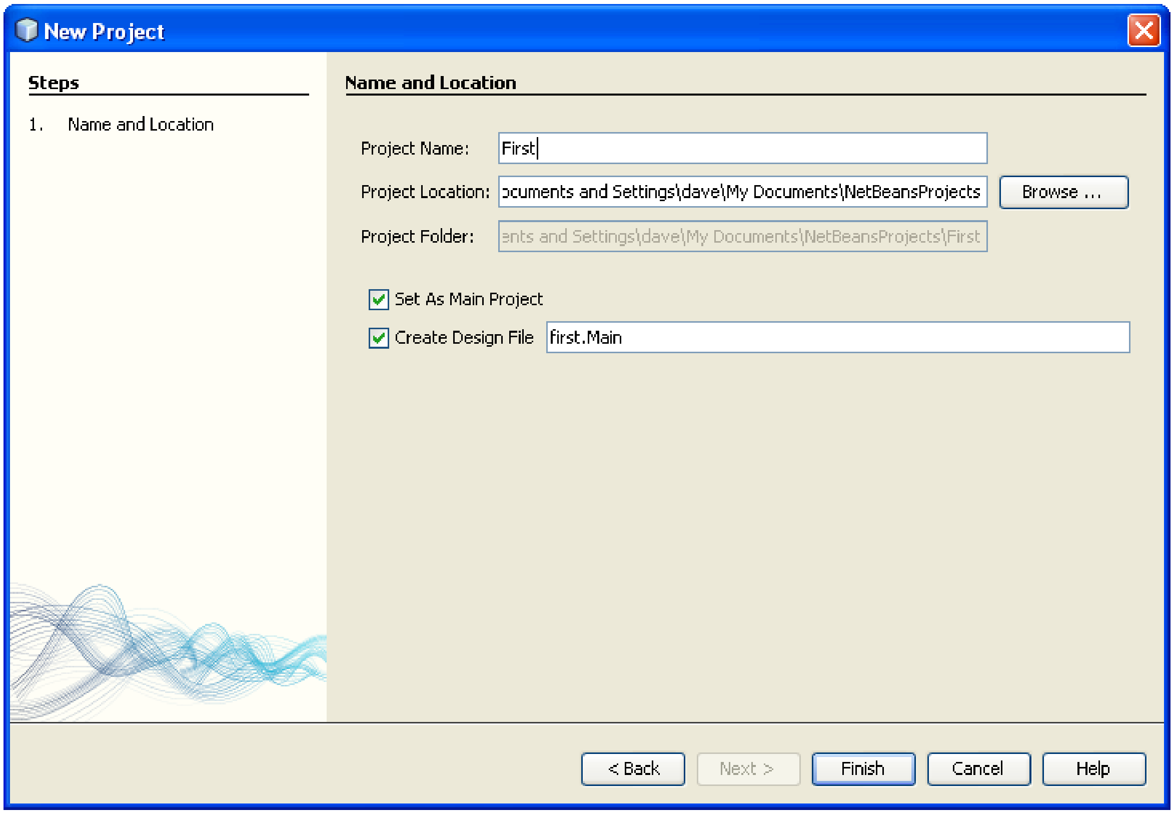
- JavaFX Script Application - creates a new project with a single "Hello World" source file.
- JavaFX Desktop Business Application - creates a new project with "Standard Execution" profile set and a single empty JavaFX Design file.
- JavaFX Mobile Business Application - creates a new project with "Run in Mobile Emulator" profile set and a single empty JavaFX Design file.



Creates a new JavaFX Desktop application in a standard IDE project. Standard projects use **an IDE-generated Ant build script** to build and run your project.

Desktop Business Application is opened in the Visual Designer and can connect to various data sources.

Select the "JavaFX Desktop Business Application" item. Press Next. The Name and Location page appears.



1. Name and Location

Name and Location

Project Name: First

Project Location: ocuments and Settings\dave\My Documents\NetBeansProjects

Project Folder: ents and Settings\dave\My Documents\NetBeansProjects\First

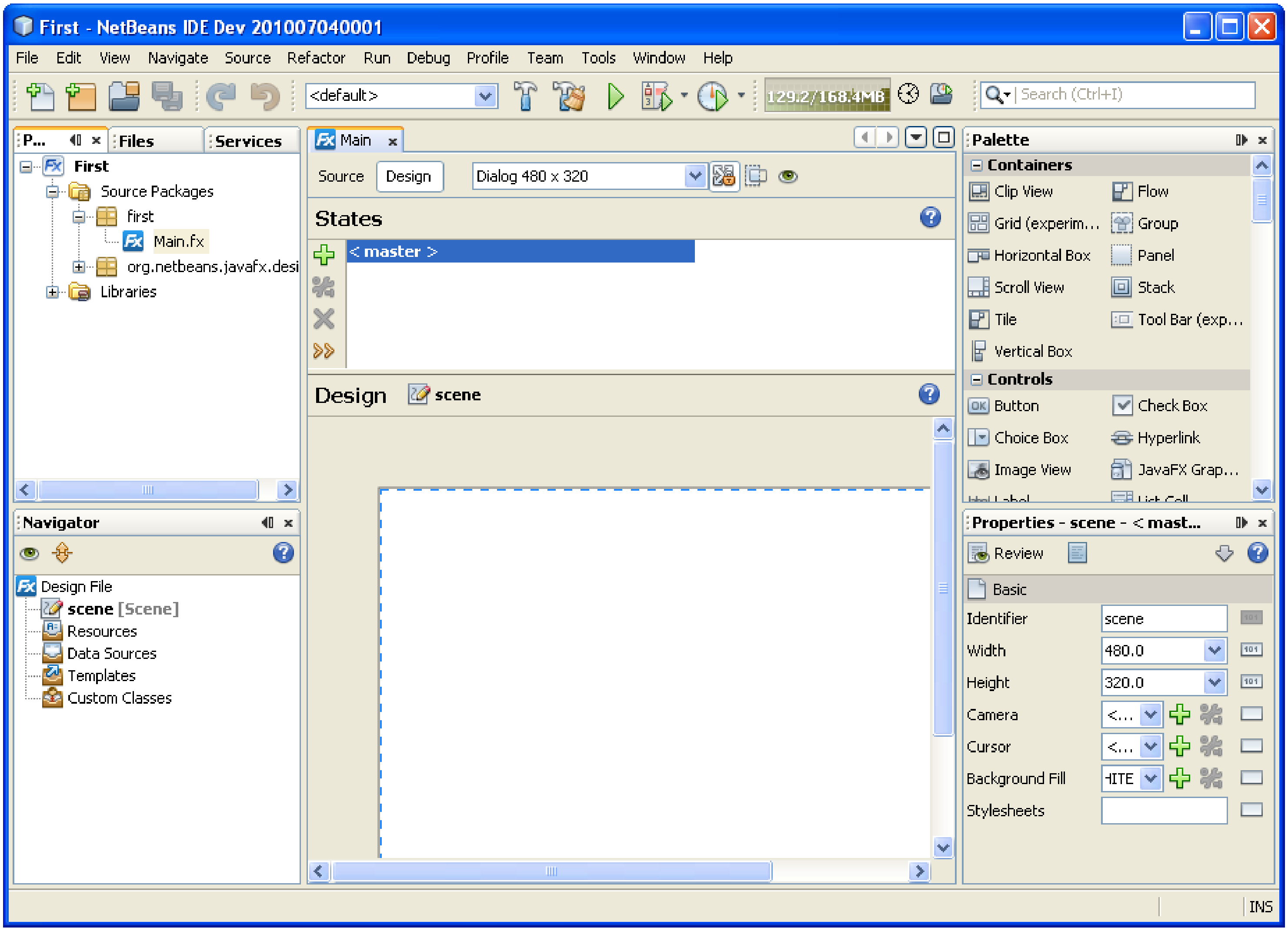
☑ Set As Main Project

☑ Create Design File first.Main

In the "Project Name"field, type "First". Press Finish.

Now the project is created and an empty design file opens in the editor area. You should see the following windows:

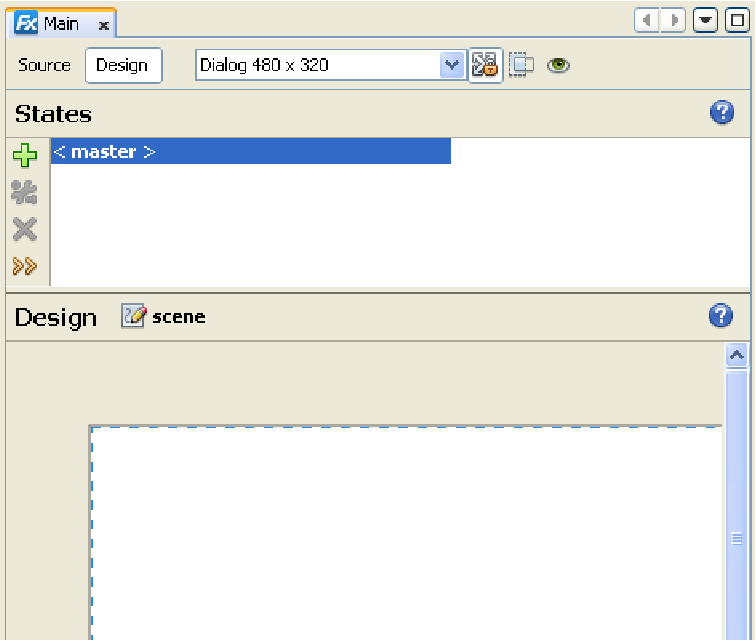
- Projects window - Logical project structure - Can be opened using "Window | Projects" main menu action.
- Navigator window - The edited design file structure - Can be opened using "Window | Navigating | Navigator" main menu action. The Navigator content is sensitive to the item currently selected in the IDE. Therefore you have to click somewhere in the Design editor to have the design file structure display properly.
- Design window - The editor of your source and design file - Can be opened by double-clicking on "First | Source Packages | first | Main.fx" node in the "Projects" window.
- Palette window - Contains all components that can be drag and dropped to your Design editor to compose your design - Can be opened using "Window | Palette" main menu action.
- JavaFX Composer Properties window - Contains property editors for currently selected components in the Design editor or Navigator - Can be opened using "Window | JavaFX Composer Properties" main menu action.



Identifier	scene
Width	480.0
Height	320.0
Camera	<...
Cursor	<...
Background Fill	HITE
Stylesheets	

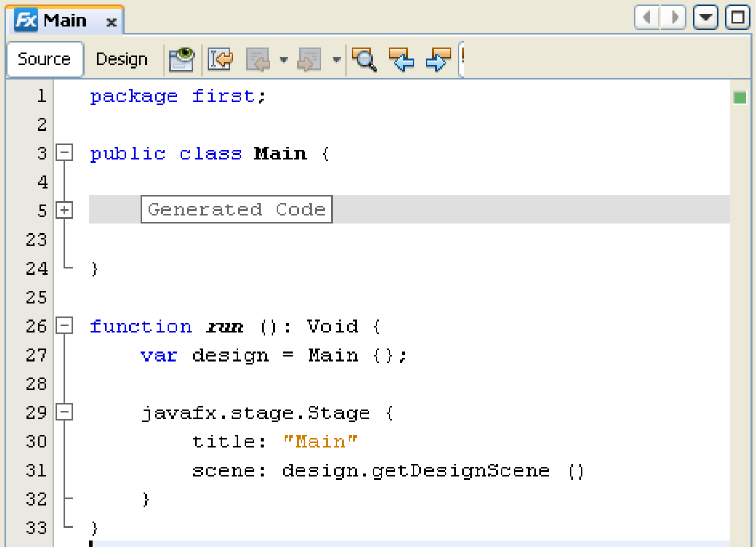
The design window has the following parts:

- Source button - Switches to the Source editor.
- Design button - Switches to the Design editor.
- The profile combo-box - Simulates the scene size for your design.
- States editor - The editor for states in your design.
- Design editor - The editor for your design. The white area represents the visible area of your scene. The gray area is a virtual space for placing your components.



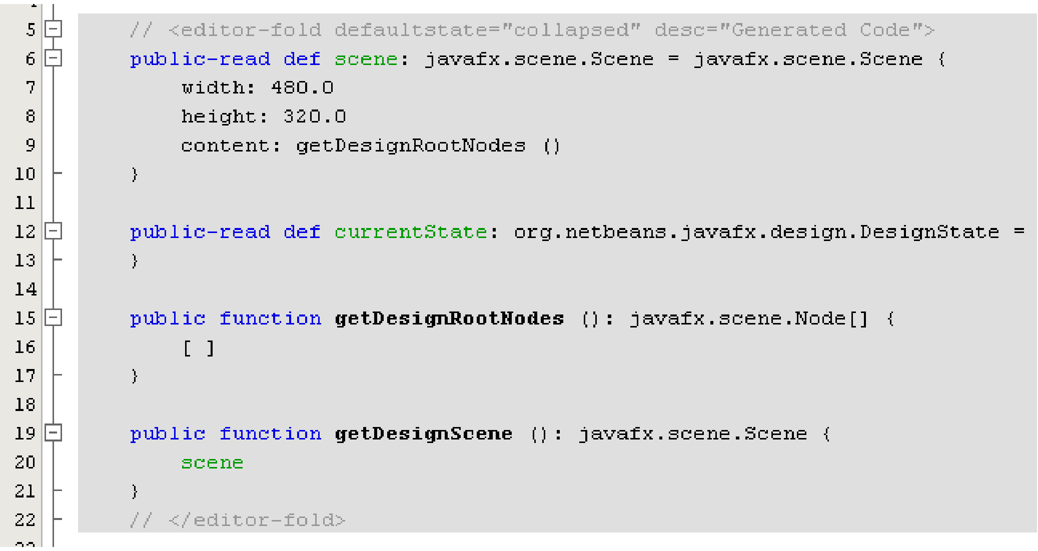
Click the "Source" button. The design window switches to the Source editor.

The source code represents your design as a JavaFX class called "Main" with a single collapsed fold - the grayed out line with "Generated Code" text and the script-level "run" function.



```
package first;

public class Main {

    Generated Code

}

function run (): Void {
    var design = Main {};

    javafx.stage.Stage {
        title: "Main"
        scene: design.getDesignScene ()
    }
}
```

Click on the "plus" icon ⊞ on the left side of the line. The fold expands. It contains the code that you have edited in the Design editor.



```
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    public-read def scene: javafx.scene.Scene = javafx.scene.Scene {
        width: 480.0
        height: 320.0
        content: getDesignRootNodes ()
    }

    public-read def currentState: org.netbeans.javafx.design.DesignState =
    }

    public function getDesignRootNodes (): javafx.scene.Node[] {
        [ ]
    }

    public function getDesignScene (): javafx.scene.Scene {
        scene
    }
    // </editor-fold>
```

The fold is regenerated every time you modify your design in the Design editor. Therefore the fold is grayed out so it cannot be modified by a developer. Note that you should not modify the grayed out code even using an external tool.

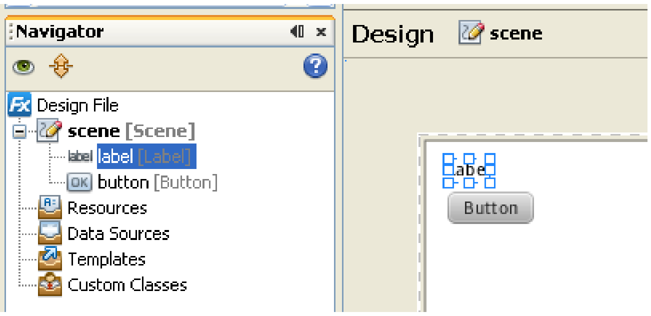
The "run" function is analogous to the "main" static method in Java. Therefore the source file acts as the start script of your application.

Editing the Design

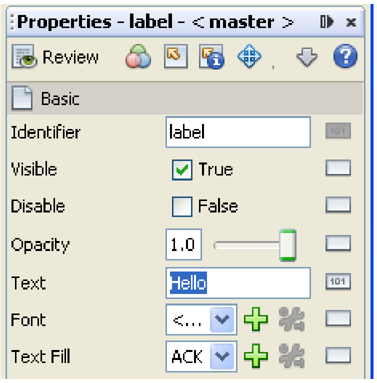
Press the "Design" button. The design window switches to the Design editor.

Scroll through the contents of the Palette window. Drag the "Label" item from the "Controls" category in the Palette window and drop it to your Design editor. Then drag and drop the "Button" item from the Palette to somewhere below the Label in the Design editor.

Select the "Label" component by clicking on it in the Design editor or by selecting the node "Design File | scene [Scene] | label [Label]" in the Navigator.



In the Properties window, set the value of the "Text" property to "Hello":



Properties - label - < master >

Review

Basic

Identifier: label

Visible: ☑ True

Disable: □ False

Opacity: 1.0

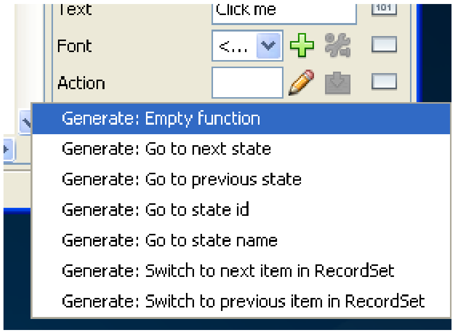
Text: Hello

Font: <...

Text Fill: ACK

Similarly select the "Button" component ("Design File | scene [Scene] | button [Button]" in Navigator). In the button component's "Text" property field, type "Click me".

The "button" component has the "Action" property to define its action when pressed. Click the "Generate" button ✎.



Text: Click me

Font: <...

Action:

Generate: Empty function
Generate: Go to next state
Generate: Go to previous state
Generate: Go to state id
Generate: Go to state name
Generate: Switch to next item in RecordSet
Generate: Switch to previous item in RecordSet

In the menu, select the "Generate: Empty function" menu item. The design window switches to the Source editor. The IDE generates a new function called "buttonAction" with an empty body. Modify the function body to look like this:

```
function buttonAction(): Void {
    label.text = "Hello World!";
}
```

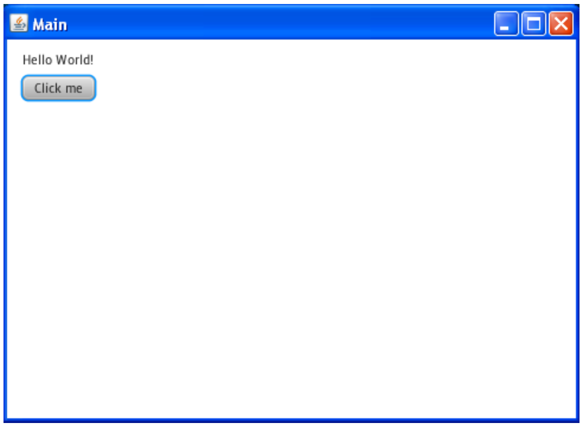
You can re-format your source code by invoking the "Source | Format" main menu action.

Click on the "plus" icon ⊞ on the left-side of the grayed out fold labeled "Generated Code" again. See that the generated code contains both the "label" and "button" components.

Running the Project

Invoke the "Run Project" pop-up menu action on the "First" project node in the Projects window.

Your application builds and starts. Press the "Click me" button in your application. The application changes appearance and now looks like this:



Main

Hello World!

Exercise 2: Application States - Master

In the exercise, we are going to create an "Image Viewer" application. We are going to use the Master/Detail UI design pattern to demonstrate how to create a dynamic application.

Introduction

So far, we can create and run an application which has a static UI. NetBeans JavaFX Composer allows you to also design a dynamic UI for your application. NetBeans JavaFX Composer's dynamic UI design function is based on the concept of States.

The term State represents the state of your application in a particular time. The state of your application is the appearance and the behavior of the UI and the current values of the application's internal properties. As your application runs, it may have multiple states. For example, you can have an application with two states:

- "Opened" state where your side-bar is expand/visible
- "Closed" state where your side-bar is collapsed/invisible

Note that your application can be in only one of these two ("Opened" and "Closed") states at one time. By switching/transitioning between these states you are actually changing the UI of your application - in this case, it is a change only of the appearance of the UI.

The NetBeans JavaFX Composer tool allows to design your application using the concept of the States described above.

The tool works with two state types:

- Single Master state
- Multiple derived states

You design always has just one Master state. If you edit your design in the Master state, you are actually editing your design as it looks immediately after the start of your application. Editing in the Master state can also be understood as editing the initial values of the properties of your components in the design.

Optionally your design can have additional "derived" states. Derived states automatically derive all values of the properties of the design components from the Master state. Additionally you are able to edit some values so they can override the value from the Master state. In each derived state, you can edit different properties. For example, you can have a design with a Label where:

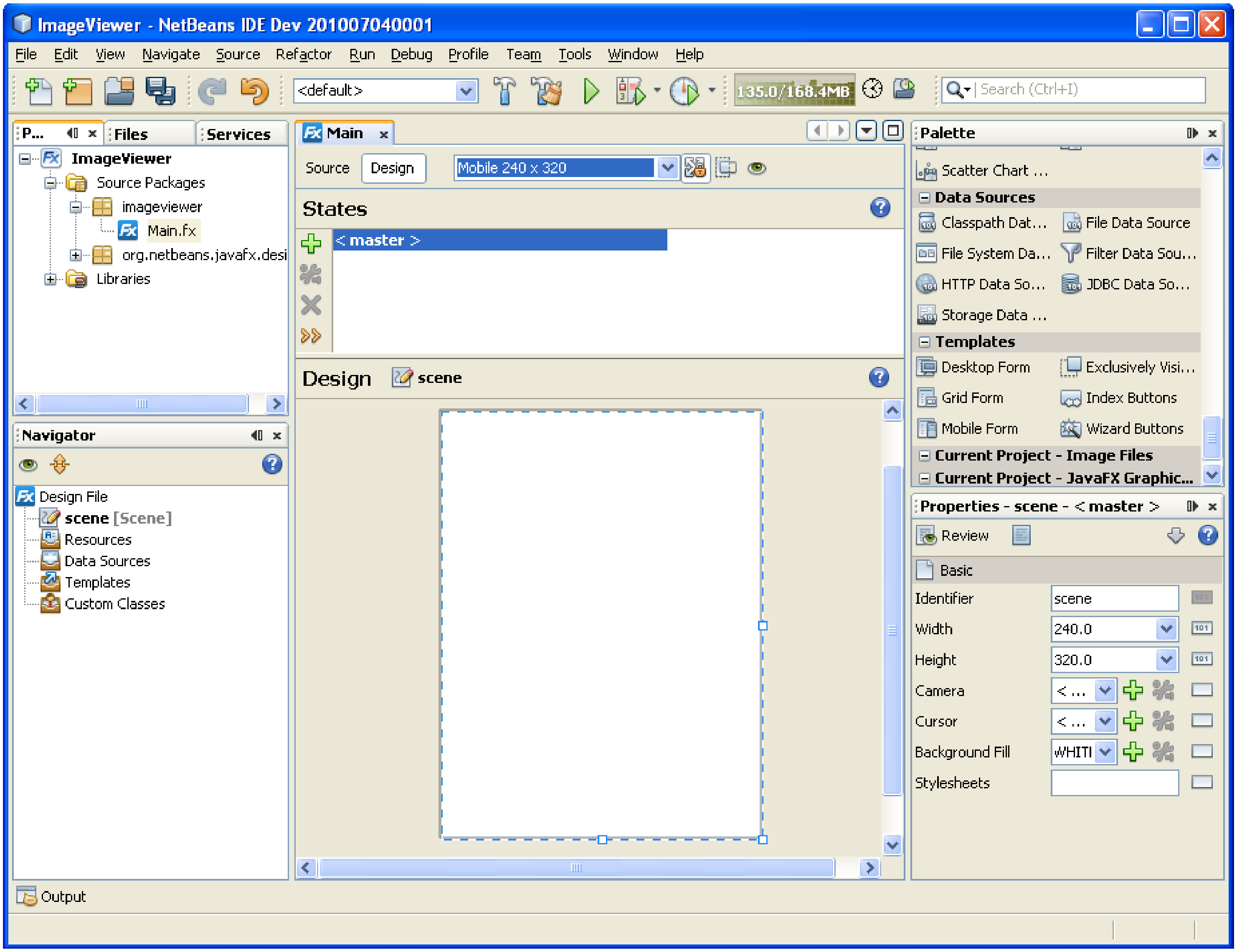
- In the Master state, the Label color is black.
- In one derived state called "MyBlueState", the Label color is blue.
- In another derived state called "MyRedState", the Label color is red.

Preparing the Design

Invoke the "File | New Project" main menu action. In the first page of the wizard, select the "JavaFX" item in the "Categories" list and "JavaFX Desktop Business Application" item in the "Projects" list. Press the "Next" button. In the second step of the wizard, type "ImageViewer" in the "Project Name" text-field. Press "Finish". The project is created and it should look similar to the "First" project.

We are going to create an application with Master/Detail design pattern. This pattern is useful for limited screen-sizes where you want to display more information.

Choose "Mobile 240x320" in the screen-size combo-box in the tool-bar of the design window. Now your design should look like this:



Identifier: scene
Width: 240.0
Height: 320.0
Camera: < ...
Cursor: < ...
Background Fill: WHITI
Stylesheets:

Our application is going to consist of two screens called "Listing" and "Detail". In the "Listing" screen, we will see a list of pre-defined image names and a "Show" button. In the "Details" screen, we will see the image itself and a "Back" button. The "Show" and "Back" buttons will guide us between the screens.

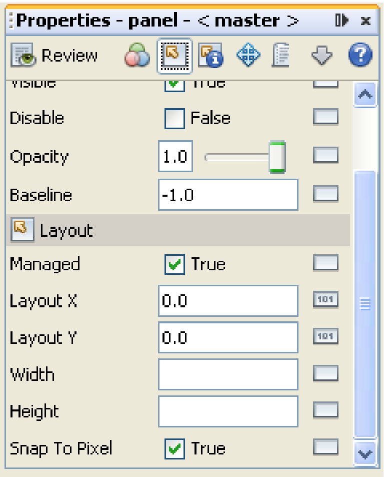
We are going to design the screens side-by-side in the Design editor. Then we are going to introduce the two states "Listing" and "Detail".

Editing in the Master State

In the Palette window, scroll up to the "Containers" category. Drag the "Panel" item and drop it in the Design editor, in the scene's left-top corner. Horizontal and vertical red guidelines appear, helping you to align the Panel.

If you do not want to use the drag and drop feature, you can add a new Panel into the Scene using the "Add | Containers | Panel" pop-up menu action on the "Design File | scene [Scene]" node in the Navigator window.

The created "panel" component should be automatically selected. Go to the Properties window. In the Properties tool-bar, press the "Layout" button . More property editors appear in the Properties window. Check if the "Layout X" and "Layout Y" properties have the value "0.0". If not, edit the layout properties to have these values. The Properties window should look like this:

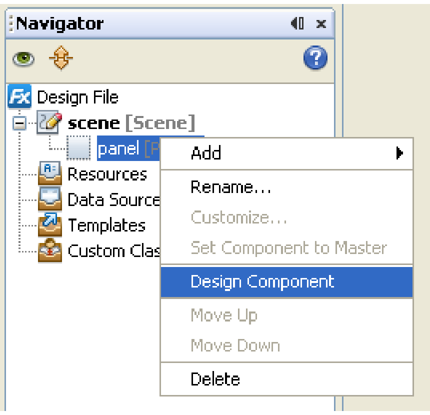


Properties - panel - < master >

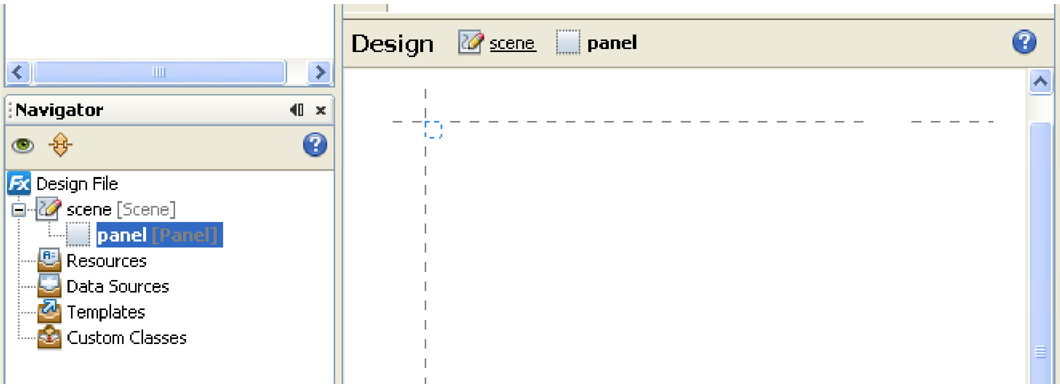
Review

Visible	☑ True
Disable	□ False
Opacity	1.0
Baseline	-1.0
Layout	
Managed	☑ True
Layout X	0.0
Layout Y	0.0
Width	
Height	
Snap To Pixel	☑ True

Invoke the "Design Component" pop-up menu action on the "Design File | scene [Scene] | panel [Panel]" node in the Navigator window.



See that the Design editor is changed and it has a white background. This means that we are editing the "panel" itself and we have unlimited space within it. The Design editor should look like this:



Drag the following items from the "Controls" category in the Palette window and drop them to the Design editor:

- One "List View" item for the images listing
- One "Button" item for the Show button
- One "Image View" item for showing an image
- One "Button" item for the Back button

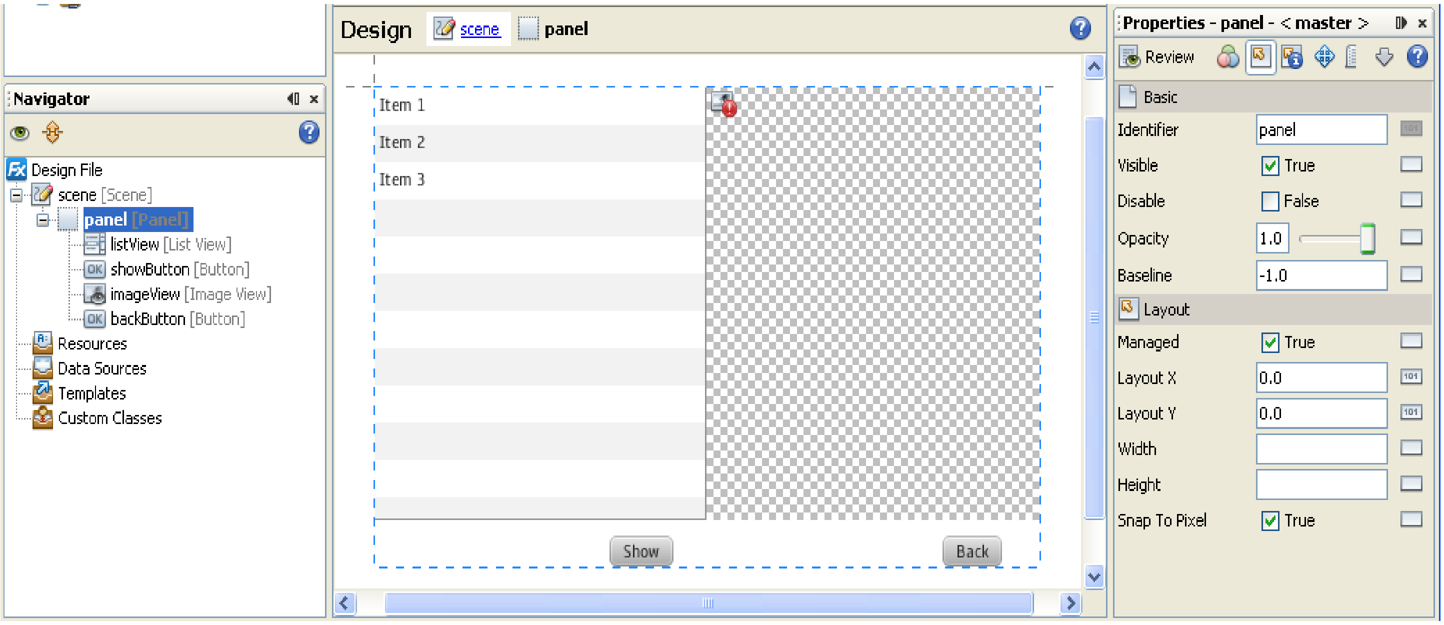
To distinguish the Buttons, select the first one and in the Properties window change its "Identifier" property to "showButton" and its "Text" property to "Show". Similarly select the second Button and edit its "Identifier" property to "backButton" and its "Text" property to "Back".

Edit the location and size of the added components:

- The "listView" component should be placed at location [0,0] . Set its "Layout X" property to "0" and "Layout Y" property to "0".
- The "listView" component should be spread across the whole screen width. Therefore set its "Width" property to "240" and "Height" property to "280".
- The "showButton" component should be placed at location [170,290]. Set its "Layout X" property to "170" and "Layout Y" property to "290".
- The "imageView" component should be placed at location [240,0]. Set its "Layout X" property to "240" and "Layout Y" property to "0".
- The "imageView" component should be spread across the whole screen width. Therefore set its "Fit Width" property to "240" and "Fit Height" property to "280".
- The "backButton" component should be placed at location [410,290]. Set its "Layout X" property to "410" and "Layout Y" property to "290".

Additionally select the "imageView" component and set its "Preserve Ratio" property to "True" by checking the check-box in the Properties window.

Now our application should look like two screens placed side-by-side.



Properties - panel - < master >

Basic

Identifier panel

Visible ☑ True

Disable □ False

Opacity 1.0

Baseline -1.0

Layout

Managed ☑ True

Layout X 0.0

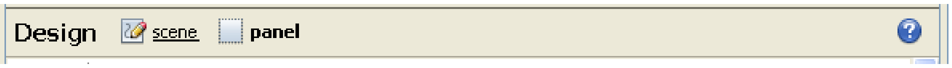
Layout Y 0.0

Width

Height

Snap To Pixel ☑ True

Click on the "scene" hyperlink on right-side of the "Design" text in the breadcrumb in the Design editor.



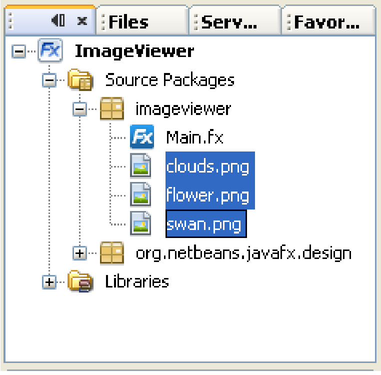
We have returned to editing the whole scene.

Adding Images

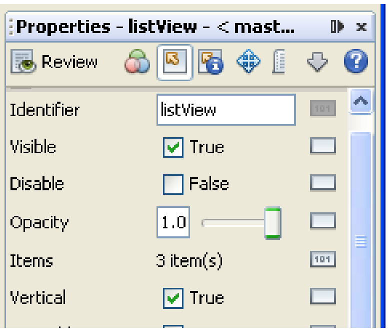
The "listView" still has some predefined values. Instead of these values, we should have the listing of our images. In your "Home" directory there are a few images prepared for you.

Invoke the "Window | Favorites" main menu action. In the "Favorites" window, expand your "Home" directory. Select "clouds.jpg", "flower.jpg", and "swan.jpg". Invoke the "Copy" pop-up menu action on one of the selected files.

Switch back to the "Projects" window. Invoke the "Paste" pop-up menu action on the "ImageViewer | Source Packages | imageviewer" package node. The image files are copied to the "imageviewer" package.

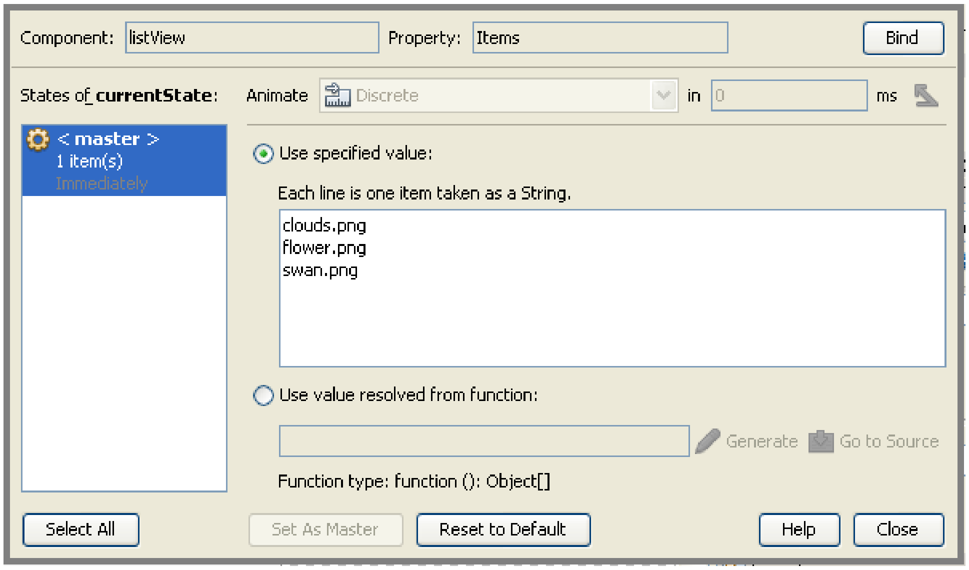


Select the "listView" component in your Design editor. Go to the Properties window. There is a property editor for the "Items" property. Press the "Details" button on the right-side of the property editor.



Identifier listView
Visible ☑ True
Disable □ False
Opacity 1.0
Items 3 item(s)
Vertical ☑ True

A "Details" window opens to allow you to edit the list of the items within the "listView" component. Delete the predefined "Item 1", "Item 2", "Item 3" lines and enter the following new ones: "clouds.png", "flower.png" and "swan.png" - each item on a new line. Now the "Details" window should look like this:



Component: listView Property: Items

States of currentState: Animate Discrete in 0 ms

< master >
1 item(s)
Immediately

● Use specified value:

Each line is one item taken as a String.

```
clouds.png
flower.png
swan.png
```

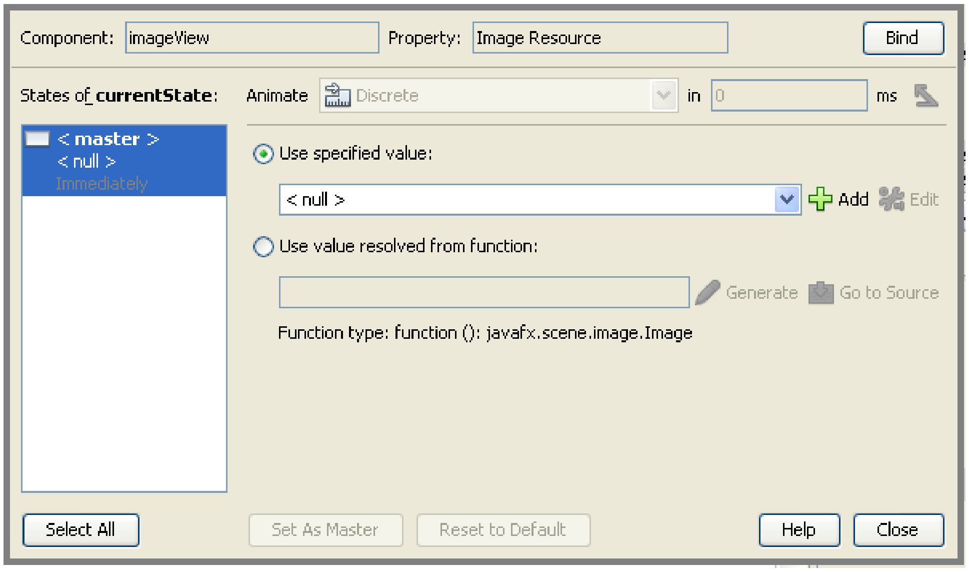
○ Use value resolved from function:

Function type: function (): Object[]

Press "Close" to close the "Details" window.

Now we have to define that the "imageView" component displays an image that is selected in the "listView" component. To do so, we are going to use JavaFX Script language feature called binding expressions. The NetBeans JavaFX Composer tool allows you to bind almost any property of any component in your design. In our case, we need to bind the "Image Resource" property of the "imageView" component.

Select the "imageView" in the Design editor. Go to the Properties window. Press the "Details" button on the right-side of the "Image Resource" property editor. A "Details" window appears.



Component: imageView

Property: Image Resource

States of currentState:

Animate: Discrete in 0 ms

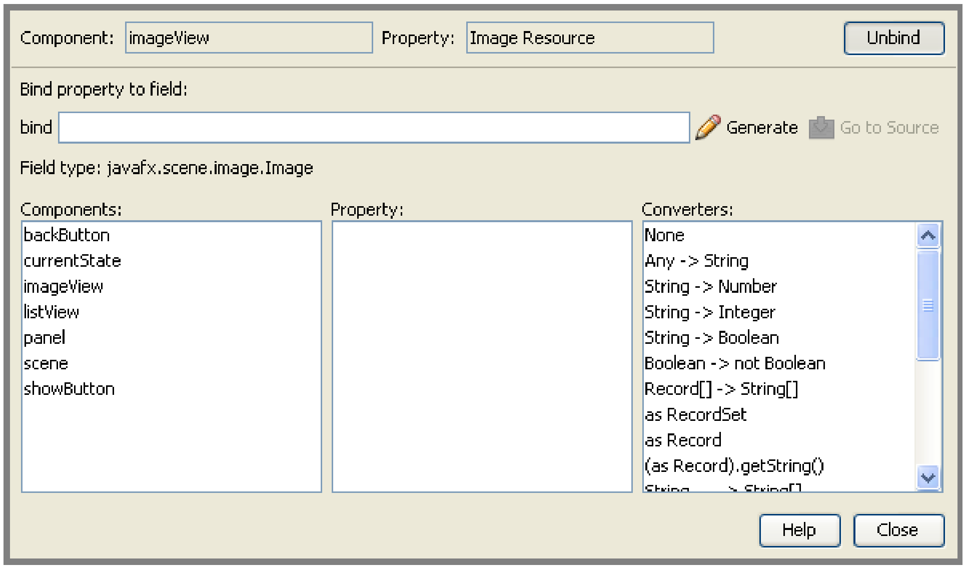
< master >
< null >
Immediately

● Use specified value: < null >

○ Use value resolved from function:

Function type: function (): javafx.scene.image.Image

Press the "Bind" button at the top-right corner of the window to switch the window to the binding mode.



Component: imageView

Property: Image Resource

Bind property to field:

bind

Field type: javafx.scene.image.Image

Components:
backButton
currentState
imageView
listView
panel
scene
showButton

Property:

Converters:
None
Any -> String
String -> Number
String -> Integer
String -> Boolean
Boolean -> not Boolean
Record[] -> String[]
as RecordSet
as Record
(as Record).getString()

In the binding mode of the window, press the "Generate" button. The design window automatically switches to the Source editor. A new variable is generated in the Source editor:

```
var imageViewImage: javafx.scene.image.Image = bind null;
```

Modify the code to look like this:

```
var imageViewImage: javafx.scene.image.Image =
    bind if (listView.selectedItem != null)
    then javafx.scene.image.Image {
        url: "{__DIR__}{listView.selectedItem}"
    }
    else null;
```

The code above creates an instance of Image resource based on the selected item in the listView. The "Image" property of the "imageView" component is bound to this variable and therefore a particular image file is shown in the "imageView".

Press "Design" button to switch back to the Design editor.

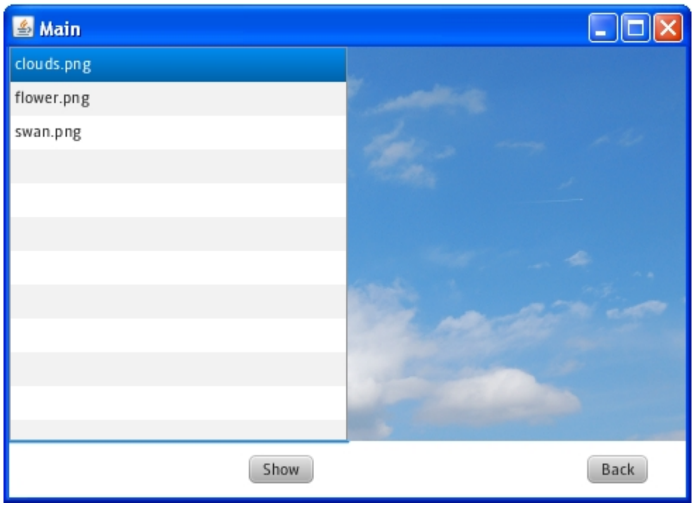
Testing the Application

Our application should be working now except for the Buttons, whose logic is not implemented yet.

To test the application, select the "Dialog 480x320" item in the screen-size combo-box in the tool-bar of the design window. Now your design should fit the whole scene.

Invoke the "Run Project" pop-up menu action on the "ImageViewer" project node in the "Projects" window.

The application is built and started using Standard Execution. You can select an image name in the List View and the image is shown on the right side. The Buttons do not do anything.



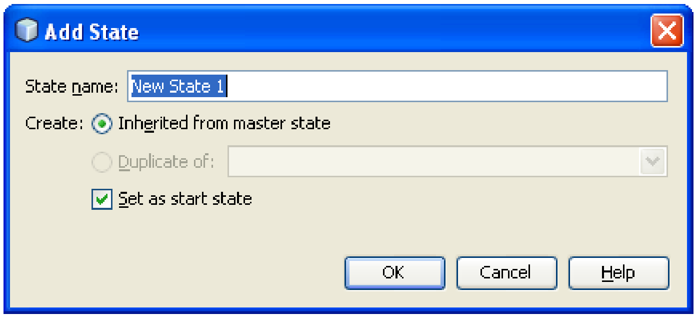
Close the application. Select the "Mobile 240x320" item in the screen-size combo-box in the tool-bar of the design window.

Exercise 3: Application States - States

In this exercise, we are going to add the states and state-specific design changes to our application.

Editing States

Go to the States editor in the Design window. Press the "Add" button ✚ in the tool-bar in the States editor. The "Add State" dialog appears.



State name: New State 1

Create: ● Inherited from master state

○ Duplicate of:

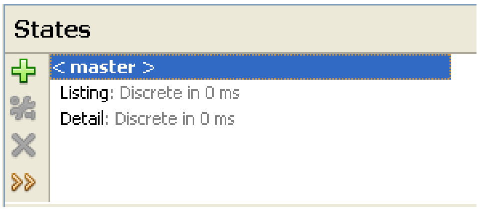
☑ Set as start state

Change the "State name" text to "Listing" and press "OK". The "Listing" state is created.

Press the "Add" button ✚ again and change the "State name" text to "Detail". Press "OK" to create the second state.

Now the States editor should contain three states:

- < master >
- Listing
- Detail



< master >

Listing: Discrete in 0 ms

Detail: Discrete in 0 ms

Click on each state in the States editor. Note that the design looks exactly the same in all states. Now it is time to modify them.

We will not modify the Master and "Listing" states since they look exactly how we want them to. We just need to modify the design in the "Detail" state so the scene will show the "imageView" part only.

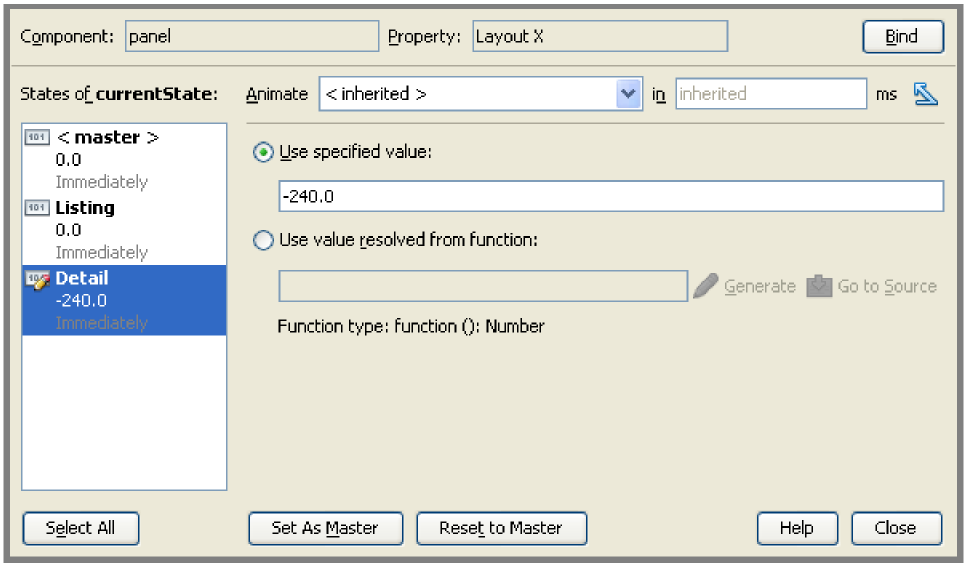
Select "Detail" state in the States editor. Select the "Design File | scene [Scene] | panel [Panel]" node in the Navigator. Go to the Properties window. Set the "Layout X" property to "-240" value.

Note that the icon of the "Details" button on the right-side of the property editor is changed to ✎.

The "Details" button has a different icon based on the value in the selected state:

- - The default value of the property is used.
- - The value of the property that was set in the Master state is used.
- - The value of the property is specified to override the default value or the value set in Master state.
- - The value of the property cannot be modified at all.

Press the "Details" button. The "Details" window is shown. It contains a list of all states of in your design and shows the property value for each state. For the "Layout X" property of the "panel" component it look like this:



Component: panel Property: Layout X

States of currentState: Animate < inherited > in inherited ms

< master >
0.0
Immediately

Listing
0.0
Immediately

Detail
-240.0
Immediately

● Use specified value:
-240.0

○ Use value resolved from function:

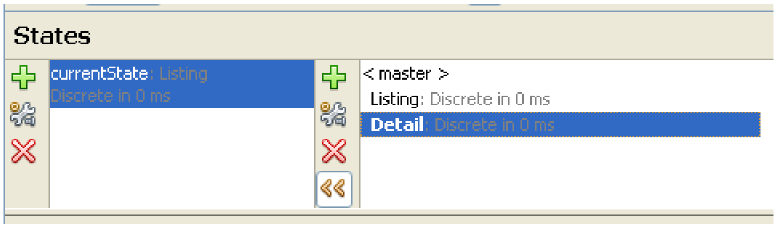
Function type: function (): Number

You can select each state and see the difference made in your design. Now our design fits to the screen size limit and each state shows a particular part of the application.

Using State Variable

Now we have to implement the switch between the states in the design.

Click the "Show state variable editor" button in the tool-bar in the States editor. Now the States editor looks like this:



States

currentState: Listing
Discrete in 0 ms

< master >
Listing: Discrete in 0 ms
Detail: Discrete in 0 ms

You see that there is a state variable named "currentState". This variable represents the current state of the design. The "Listing" state is automatically set as the start state in the design. This means that right after the application is started and the initial values of all properties are set based on the design in the Master state, the application is immediately switched to the "Listing" state. In our case, the Master and "Listing" states looks exactly the same, so you do not see any difference when you start the application.

Now let's implement the logic of the buttons.

Select the Master state in the States editor. Select the "Show" button in the Design editor. Go to the Properties window. Press the "Generate" button ✎ in the "Action" property editor. A pop-up menu opens. In the pop-up menu, select the "Generate: Go to state name" item. The design window immediately switches to the Source editor. Then a new function is created with the pre-created body:

```
function detailButtonAction(): Void {
    currentState.actual = currentState.findIndex("theStateName");
}
```

Press the Enter key. The live template moves you to edit the "theStateName" text. Replace the text with "Detail", which is the name of the state that you want the application to switch to. Now the code looks like this:

```
function detailButtonAction(): Void {
    currentState.actual = currentState.findIndex("Detail");
}
```

Press the "Design" button to switch back to the Design editor. Select the "Back" button. Go to the Properties window and see the "Action" property editor. Press the "Generate" button ✎. In the pop-up menu, select the "Generate: Go to state name" item. The design window immediately switches to the Source editor. Then a new function is created with the pre-created body:

```
function backButtonAction(): Void {
    currentState.actual = currentState.findIndex("theStateName");
}
```

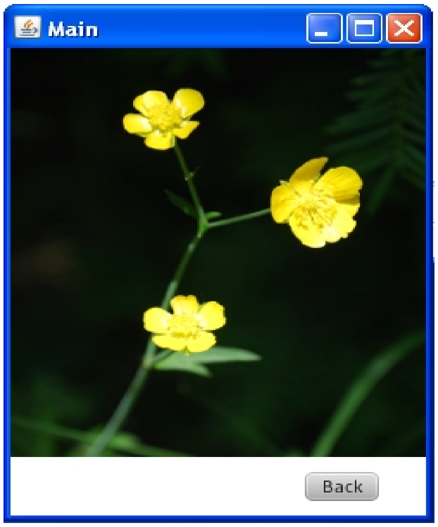
Press the Enter key so the live-template moves you to edit the "theStateName" text. Replace the text with "Listing". Now the code looks like this:

```
function backButtonAction(): Void {
    currentState.actual = currentState.findIndex("Listing");
}
```

Running the Application

Invoke the "Run Project" pop-up menu action on the "ImageViewer" project node in the "Projects" window.

The application is built and started. You can select an image in the List View, press the "Show" button and your application will show the image.

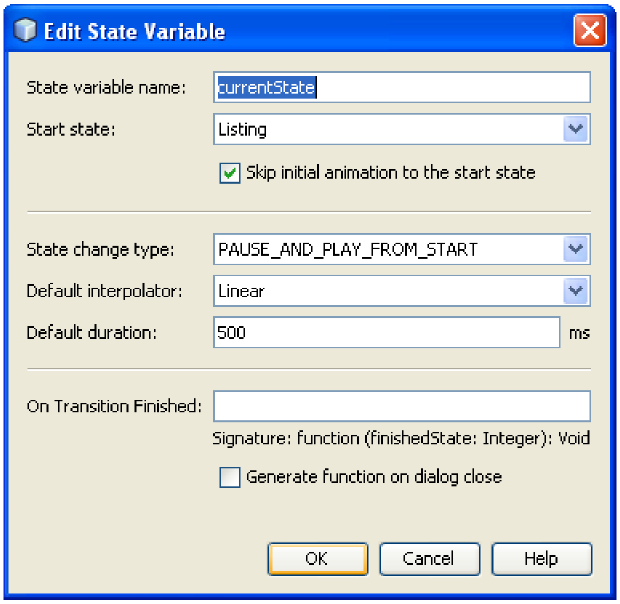


Simple Animation Support

The application switches between the states immediately after a Button is pressed. With NetBeans JavaFX Composer tool you can enhance the design with simple animations. NetBeans JavaFX Composer allows you to specify a duration and an interpolation for the animation of the transition to a target state.

In the State Variable editor, select the "currentState" item and press the "Edit" button in the tool-bar of the State Variable editor. A dialog appears.

Change the value of the "Default interpolator" combo-box to "Linear" and change the value of the "Default duration" text-field to "500". The dialog should look like this:



Edit State Variable

State variable name: currentState

Start state: Listing

☑ Skip initial animation to the start state

State change type: PAUSE_AND_PLAY_FROM_START

Default interpolator: Linear

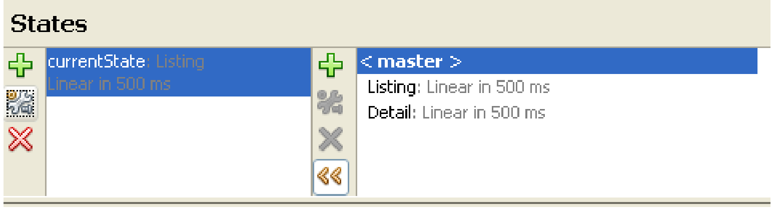
Default duration: 500 ms

On Transition Finished:

Signature: function (finishedState: Integer): Void

□ Generate function on dialog close

Press "OK" to close the dialog. Note that the states editor now lists all the states with the "Linear in 500 ms" comment, which described the default animation used for the transition into the particular state.



States

currentState: Listing
Linear in 500 ms

< master >
Listing: Linear in 500 ms
Detail: Linear in 500 ms

Note that similarly you can override the default animation of each state independently by editing a particular state.

Note that you can override the default animation for each property of each component in the design independently for each state. To override the default animation of a property, click that property's "Open Details" button in the Properties window. The property's "Details" window opens. In the "Details" window, select the state or states in which you want to change the animation, and then change the property's animation.

Running the Animation

Invoke the "Run Project" pop-up menu action on the "ImageViewer" project node in the "Projects" window.

The application is built and started. Select an image in the List View and press the "Show" button. Now your scene smoothly slides to the left so the Image View smoothly replaces the List View.

Exercise 4: Data Source

The NetBeans JavaFX Composer tool provides the ability to gather data from any support source and supply them to your application in a tree-like structure.

In this exercise, we are going to gather images using the Yahoo! Search Web Service for searching images. The gathered images are then listed and shown in the application.

The service requires to use an Application ID. We have one generated for this HOL:

```
PigYsoHV34FfXCVua8VFP0a.YWCQ1l6dq_tBPYGn0PjYEmZd9RHVU8RW8zNWcY_UBIKO
```

More information about the service can be found at:

- http://developer.yahoo.com/search/
- http://developer.yahoo.com/search/image/V1/imageSearch.html
- http://developer.yahoo.com/search/rest.html

Introduction

The Data Source feature allows an application to gather data from various source:

- JDBC Data Source - Data are taken from a database by running a select command over a JDBC driver.
- HTTP Data Source - Data are taken from a Web Service or REST service.
- File Data Source - Data are read from a file on your local file system
- Classpath Data Source - Data are read from a file within on your classpath
- Storage Data Source - Data are read from a file using Storage API
- Filesystem Data Source - Data represents a directory structure on your local file system.

No matter which Data Source you use, data are read and parsed and a tree-like structure is created. The structure consists of the following two types:

- RecordSet - represents a sequence of Record instances
- Record - represents a single Record as a sequence of attributes. Each attribute has a name (of String type) and a value. The value can be Integer, Number, String, Record, RecordSet or any Object in general.

The Data Source feature also provides meta-data for the data in the run-time so you can inspect your tree-like structure while your application is running.

Addtionally the tree-like structure can be filtered by an expression (of String type) which allows you to extract particular sub-set of data in case you are interest in a specific part of the data returned by a Data Source.

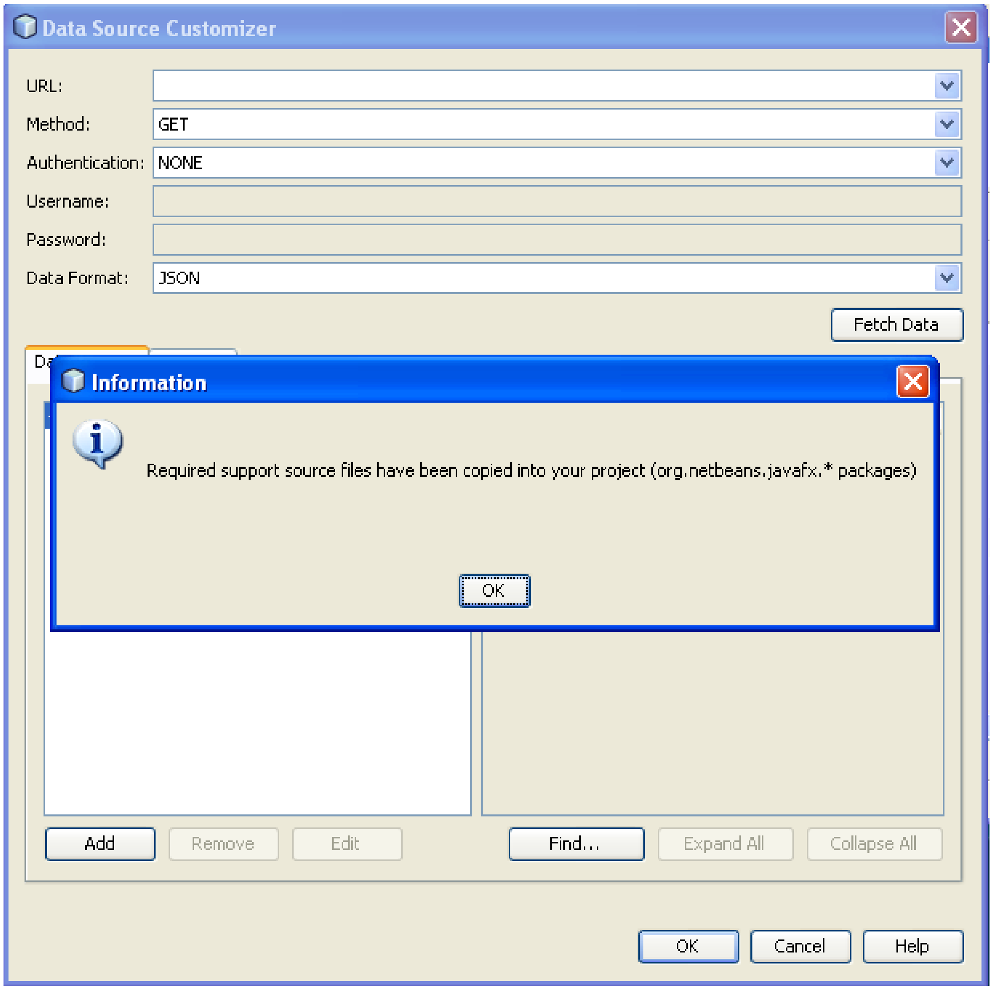
Adding HTTP Data Source

In this exercise, we are going to modify the design on the Master state only.

Switch to the Master state.

Go to the Palette and scroll to the bottom. Drag an "HTTP Data Source" item from the "Data Sources" category in the Palette and drop it somewhere in the Design editor.

Two dialogs appears. One dialog is for customizing the HTTP Data Source. The other dialog notifies you that the required JavaFX Script source files have been copied into your JavaFX project so you can work with the Data Source API.



Data Source Customizer

URL:
Method: GET
Authentication: NONE
Username:
Password:
Data Format: JSON

Information

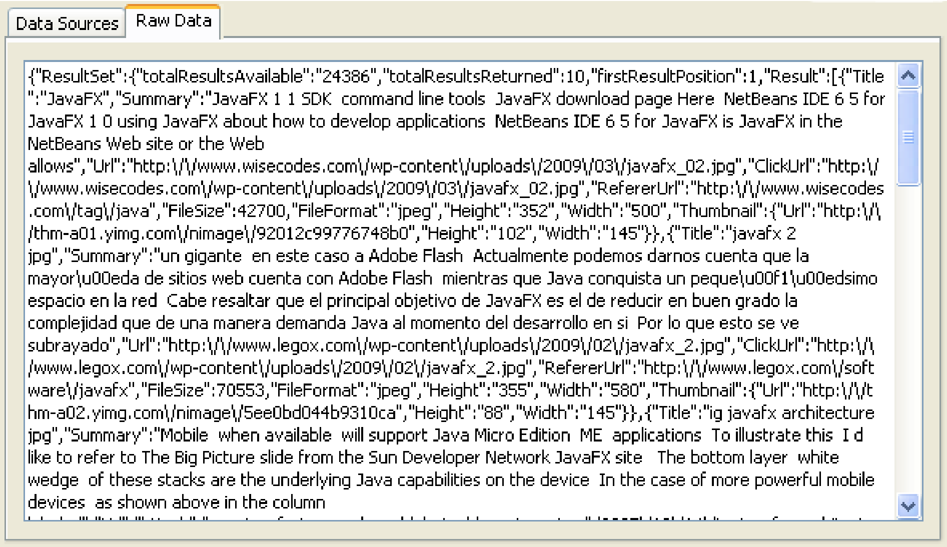
Required support source files have been copied into your project (org.netbeans.javafx.* packages)

Press the "OK" button to close the notification dialog. Now you see only the customizer. The customizer allows you to specify various attributes required for the connection.

Enter http://search.yahooapis.com/ImageSearchService/V1/imageSearch?output=json&appid=PigYsoHV34FfXCVua8VFP0a.YWCQ1l6dq_tBPYGn0PjYEmZd9RHVU8RW8zN

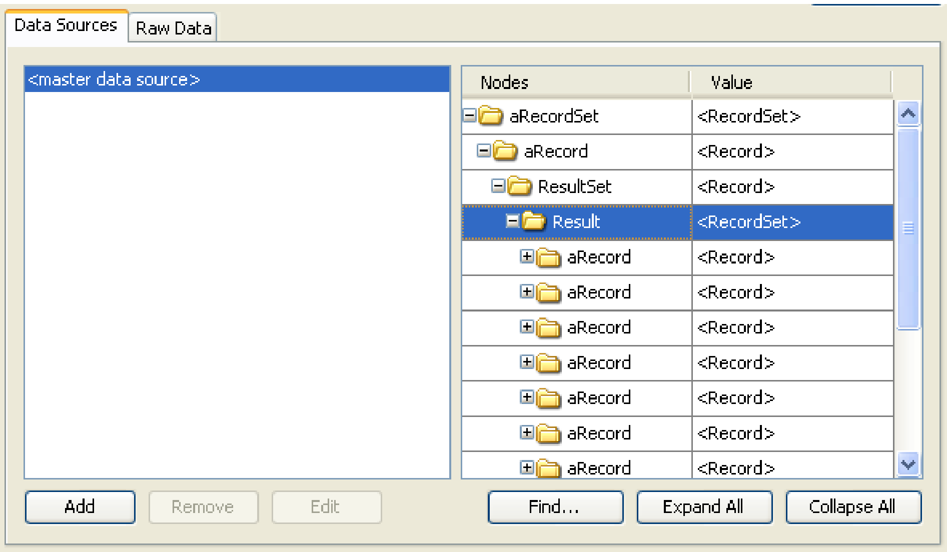
WcY_UBIKO&query=JavaFX into the “URL” text-field. Leave the other fields as they are. Press the "Fetch Data" button. Now the dialog connects to the Web Service and returns the answer displayed in the data below.

Switch to the "Raw Data" tab. The bottom part of the dialog shows the answer from the Web Service:



```
{"ResultSet":{"totalResultsAvailable":"24386","totalResultsReturned":10,"firstResultPosition":1,"Result":[{"Title
":"JavaFX","Summary":"JavaFX 1 1 SDK  command line tools  JavaFX download page Here  NetBeans IDE 6 5 for
JavaFX 1 0 using JavaFX about how to develop applications  NetBeans IDE 6 5 for JavaFX is JavaFX in the
NetBeans Web site or the Web
allows","Url":"http:\/\/www.wisecodes.com\/wp-content\/uploads\/2009\/03\/javafx_02.jpg","ClickUrl":"http:\/
\/www.wisecodes.com\/wp-content\/uploads\/2009\/03\/javafx_02.jpg","RefererUrl":"http:\/\/www.wisecodes
.com\/tag\/java","FileSize":42700,"FileFormat":"jpeg","Height":"352","Width":"500","Thumbnail":{"Url":"http:\/\
/thm-a01.yimg.com\/nimage\/92012c99776748b0","Height":"102","Width":"145"}},{"Title":"javafx 2
jpg","Summary":"un gigante  en este caso a Adobe Flash  Actualmente podemos darnos cuenta que la
mayor\u00eda de sitios web cuenta con Adobe Flash  mientras que Java conquista un peque\u00f1\u00edsimo
espacio en la red  Cabe resaltar que el principal objetivo de JavaFX es el de reducir en buen grado la
complejidad que de una manera demanda Java al momento del desarrollo en si  Por lo que esto se ve
subrayado","Url":"http:\/\/www.legox.com\/wp-content\/uploads\/2009\/02\/javafx_2.jpg","ClickUrl":"http:\/\
/www.legox.com\/wp-content\/uploads\/2009\/02\/javafx_2.jpg","RefererUrl":"http:\/\/www.legox.com\/soft
ware\/javafx","FileSize":70553,"FileFormat":"jpeg","Height":"355","Width":"580","Thumbnail":{"Url":"http:\/\/t
hm-a02.yimg.com\/nimage\/5ee0bd044b9310ca","Height":"88","Width":"145"}},{"Title":"ig javafx architecture
jpg","Summary":"Mobile  when available  will support Java Micro Edition  ME  applications  To illustrate this  I d
like to refer to The Big Picture slide from the Sun Developer Network JavaFX site   The bottom layer  white
wedge  of these stacks are the underlying Java capabilities on the device  In the case of more powerful mobile
devices  as shown above in the column
```

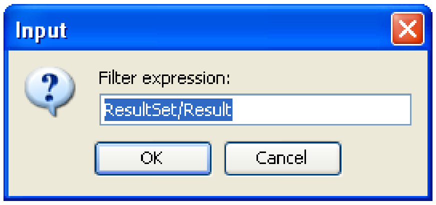
Switch back to the "Data Sources" tab. It contains a list and a tree-table. Go to the tree table and select the "aRecordSet | aRecord | ResultSet | Result" nodes. The dialog should look like this:



Nodes	Value
⊟ aRecordSet	<RecordSet>
⊟ aRecord	<Record>
⊟ ResultSet	<Record>
⊟ Result	<RecordSet>
⊞ aRecord	<Record>
⊞ aRecord	<Record>
⊞ aRecord	<Record>
⊞ aRecord	<Record>
⊞ aRecord	<Record>
⊞ aRecord	<Record>
⊞ aRecord	<Record>

The tree-table shows the data from the "Raw Data" tab parsed into a tree structure. You can see that the actual image results of your search are stored in Record instances under the "aRecordSet | aRecord | ResultSet | Result" node. This is what we are interested in, so we have to filter the data.

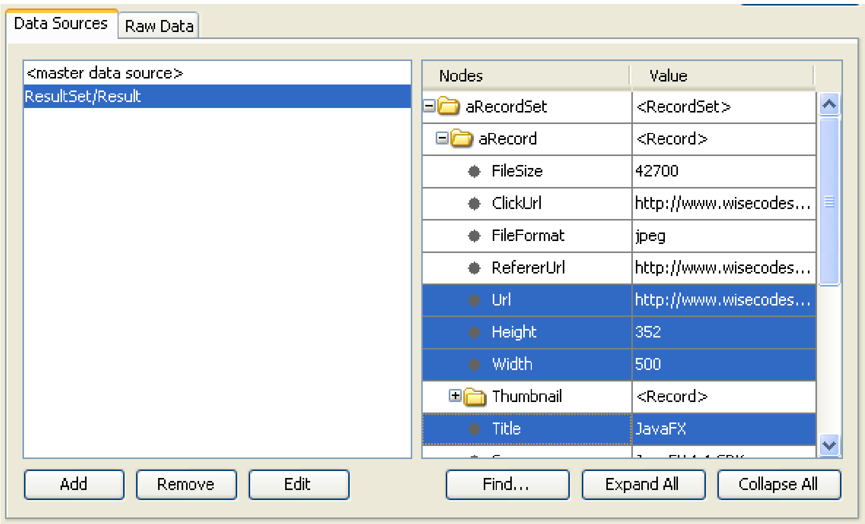
Select the "aRecordSet | aRecord | ResultSet | Result" node in the tree-table. Press the "Add" button. A dialog appears:



Filter expression: ResultSet/Result

Leave the pre-filled filter expression unmodified and press "OK". A new Data Source is defined. See the list of the left side of the dialog. There are two items:

- Master data source - Contains the tree-structure of the whole answer from the service.
- "ResultSet/Result" - Contains a RecordSet with a sequence of Record instances where each Record represents a single image that was found.



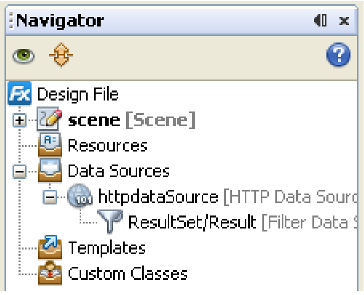
<master data source>
ResultSet/Result

Nodes	Value
aRecordSet	<RecordSet>
aRecord	<Record>
FileSize	42700
ClickUrl	http://www.wisecodes...
FileFormat	jpeg
RefererUrl	http://www.wisecodes...
Url	http://www.wisecodes...
Height	352
Width	500
Thumbnail	<Record>
Title	JavaFX

Expand the Records in the "ResultSet/Result" data source. See that each Record instance contains the following attributes. We will use these attributes later.

- Title - The title of an image
- Url - The URL of an image
- Width - The width of an image
- Height - The height of an image

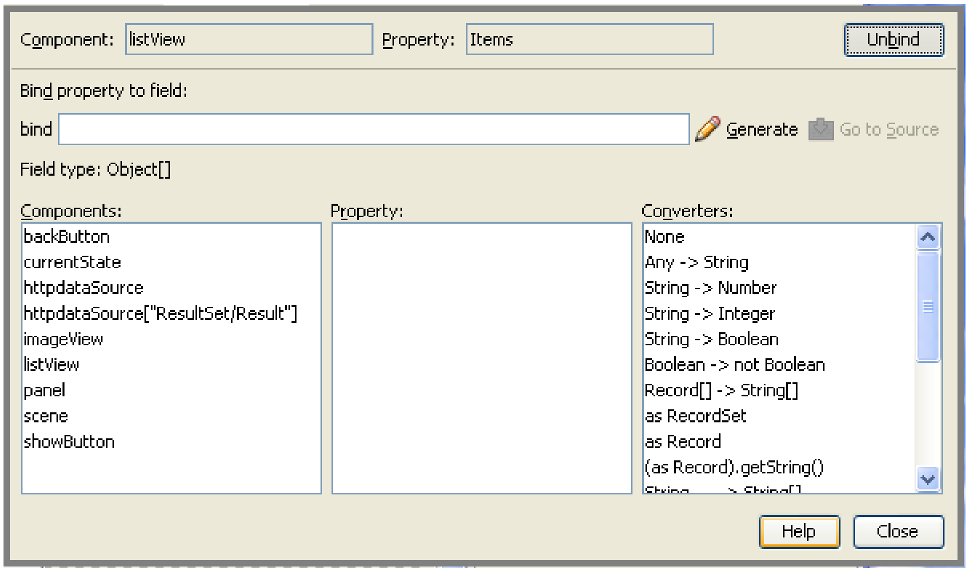
Press "OK" to close the customizer dialog. Now your design contains two Data Sources as described above but they are not connected with the rest of the application.



Connecting to the List View

Select the "clouds.jpg", "flower.jpg", and "swan.jpg" files in the "ImageViewer | Source Packages | imageviewer" package node in the "Projects" view. Invoke the "Delete" pop-up menu action on one of the selected nodes. A confirmation dialog appears. Press "Yes" in the dialog to confirm the deletion.

Go to the Design editor, select the Master state and select the "listView" component. Go to the Properties window and press the "Details" button for the "Items" property. In the "Details" window, press "Bind" to switch to the Bind mode. The window should look like this:


Component: listView Property: Items

Bind property to field:

bind

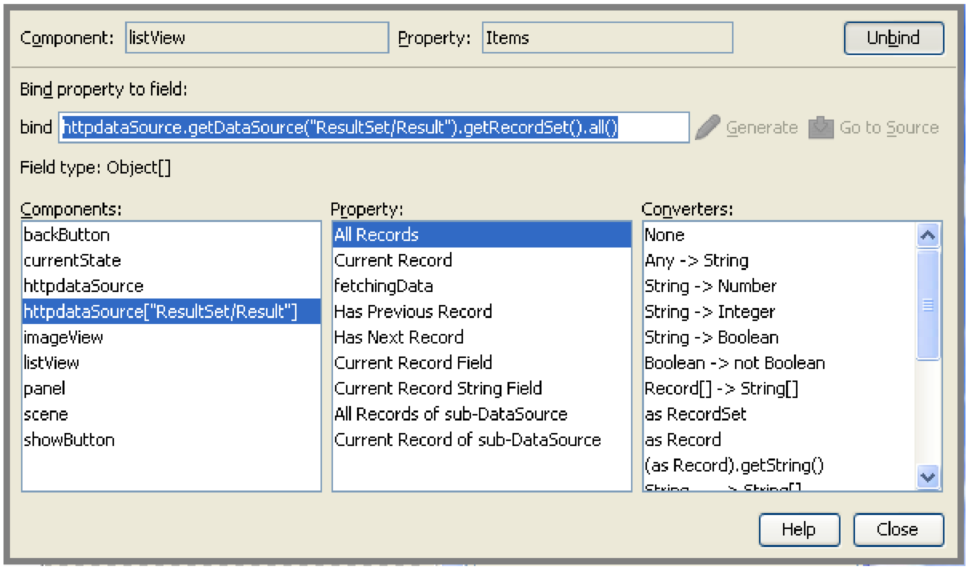
Field type: Object[]

Components:
backButton
currentState
httpdataSource
httpdataSource["ResultSet/Result"]
imageView
listView
panel
scene
showButton

Property:

Converters:
None
Any -> String
String -> Number
String -> Integer
String -> Boolean
Boolean -> not Boolean
Record[] -> String[]
as RecordSet
as Record
(as Record).getString()

In the "Details" window, select "httpDataSource["ResultSet/Result"] in the left-most list. Select "All Records" in the middle list. Note that the text-field is filled with the binding expression. The window should look like this:


Component: listView

Property: Items

Bind property to field:

bind httpdataSource.getDataSource("ResultSet/Result").getRecordSet().all()

Field type: Object[]

Components:
- backButton
- currentState
- httpdataSource
- httpdataSource["ResultSet/Result"]
- imageView
- listView
- panel
- scene
- showButton

Property:
- All Records
- Current Record
- fetchingData
- Has Previous Record
- Has Next Record
- Current Record Field
- Current Record String Field
- All Records of sub-DataSource
- Current Record of sub-DataSource

Converters:
- None
- Any -> String
- String -> Number
- String -> Integer
- String -> Boolean
- Boolean -> not Boolean
- Record[] -> String[]
- as RecordSet
- as Record
- (as Record).getString()
- String ... -> String[]

The expression looks like this:

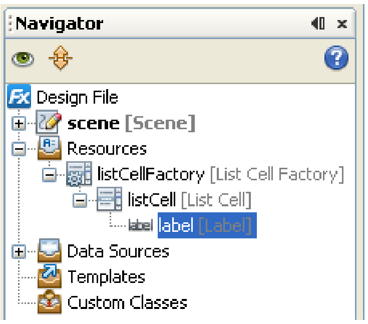
```
httpdataSource.getDataSource("ResultSet/Result").getRecordSet().all()
```

Now the "listView" component contains a sequence of Record instances where each Record instance presents a single image found by the service.

Press the "Close" button to close the window.

Since the "listView" now renders each Record instance as a cryptic string, we need to specify a "Cell Factory" to customize the look for each item in a List View.

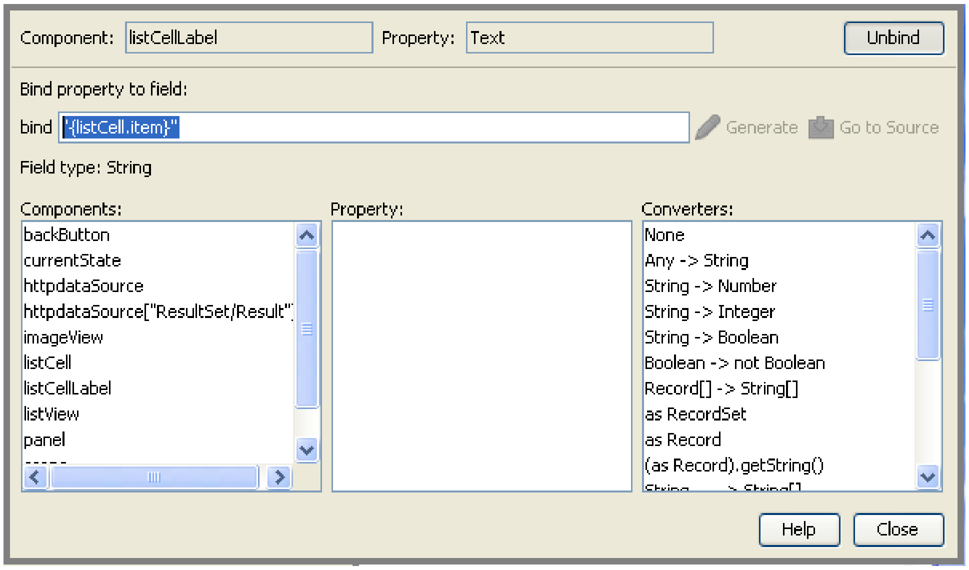
Select the Master state. Select the "listView" component in the Design editor and go to the Properties window again. Scroll to the "Cell Factory" property editor. Press the "Add new" button ✚ to create and assign a new cell factory for your "listView" component. Go to the Navigator and you should see the following structure created:


Navigator

```
Design File
⊞ scene [Scene]
⊟ Resources
   ⊟ listCellFactory [List Cell Factory]
      ⊟ listCell [List Cell]
           label [Label]
⊞ Data Sources
  Templates
  Custom Classes
```

Select the "Design File | Resources | listCellFactory [List Cell Factory] | listCell [List Cell] | label [Label]" node in the Navigator. Go to the Properties window. Change the value of the "Identifier" property to "listCellLabel".

Press the "Details" button of the "Text" property of the "listCellLabel" component. The "Details" window appears in the Bind mode. The binding expression is preset to `"{listCell.item}"`. Select the whole binding expression text.



Component: listCellLabel Property: Text

Bind property to field:

bind "{listCell.item}"

Field type: String

Components:
- backButton
- currentState
- httpdataSource
- httpdataSource["ResultSet/Result"]
- imageView
- listCell
- listCellLabel
- listView
- panel

Property:

Converters:
- None
- Any -> String
- String -> Number
- String -> Integer
- String -> Boolean
- Boolean -> not Boolean
- Record[] -> String[]
- as RecordSet
- as Record
- (as Record).getString()

Now we are going to change the binding of the "Text" of the "listCellLabel" component to show the "Title" attribute value as the text of the particular item in the "listView".

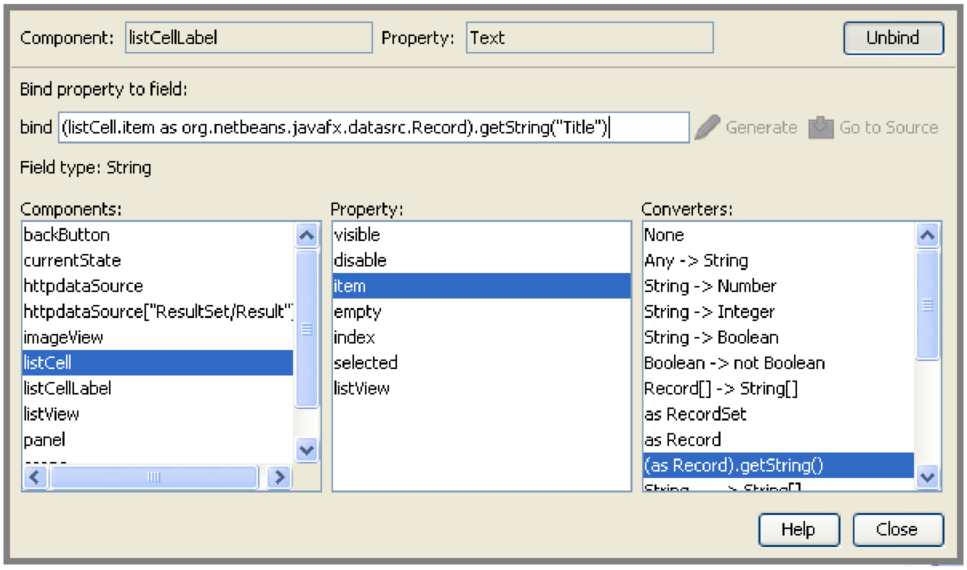
Select the "listCell" item in the Components (left-most) list. A set of Properties appears in the middle list. Select "item" in the middle list. Select the "(as Record).getString()" converter in the right-most list. The bind expression is changed and should look like this:

```
(listCell.item as org.netbeans.javafx.datasrc.Record).getString("name")
```

Note that this expression takes the "listCell.item" with the actual item to be rendered in the list. Then it is cast to a Record type and then a "getString" function is called on it to return a value of the "name" attribute. In your Record, there is no "name" attribute. Instead we would like to use value of "Title" attribute.

Manually edit the code in the text-field so the name of the attribute is changed to "Title". Now the bind expression looks like this:

```
(listCell.item as org.netbeans.javafx.datasrc.Record).getString("Title")
```



Component: listCellLabel

Property: Text

Bind property to field:

bind (listCell.item as org.netbeans.javafx.datasrc.Record).getString("Title")

Field type: String

Components: listCell

Property: item

Converters: (as Record).getString()

Press "Close" to close the "Details" window. Now the "listView" is bound to the data source and renders the image names correctly.

Connecting to the Image View

We have to modify the "imageView" component - it cannot take the currently selected item in the "listView" as an image file name (a String type).

Press the "Source" button to switch to the Source editor. Find the following code:

```
var imageViewImage: javafx.scene.image.Image =
    bind if (listView.selectedItem != null)
    then javafx.scene.image.Image {
        url: "{__DIR__}{listView.selectedItem}"
    }
    else null;
```

And modify it to look like this:

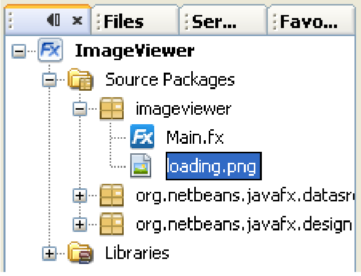
```
var imageViewImage: javafx.scene.image.Image =
    bind if (listView.selectedItem != null)
    then javafx.scene.image.Image {
        url: (listView.selectedItem as
org.netbeans.javafx.datasrc.Record).getString("Url")
        backgroundLoading: true
        placeholder: placeHolderImage
    }
    else null;
```

Now the "listView.selectedItem" is taken as a Record and its "Url" attribute is taken as the URL of the image.

We are going to use background loading for the image and we are using "placeHolderImage" image resource as a place holder. But the "placeHolderImage" does not exists yet.

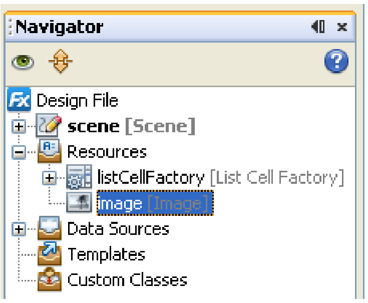
Go to the "Favorites" window. In the window, expand your "Home" directory, select "loading.png" file and invoke "Copy" pop-up menu action on the file.

Go back to the "Projects" window. Select "ImageViewer | Source Packages | imageviewer" and invoke the "Paste" pop-up menu action on the selected node. The image file is copied to the "imageviewer" package. Check that the image file is really copied to the "imageviewer" package otherwise the application will not be able to find your image. Now your project should contain this:



Go to the design window and press "Design" to switch to the Design editor again.

Go to the Palette and scroll to the bottom. There you should see the "loading.png" item in the "Current Project - Image Files" category. Drag the "loading.png" item and drop it on the "Design File | Resources" node in the Navigator. Do NOT drop it into the Design editor otherwise it is added as a "Image View" component instead of an "Image" resource. A new "image" resource should be created. The structure in the Navigator window should look like this:

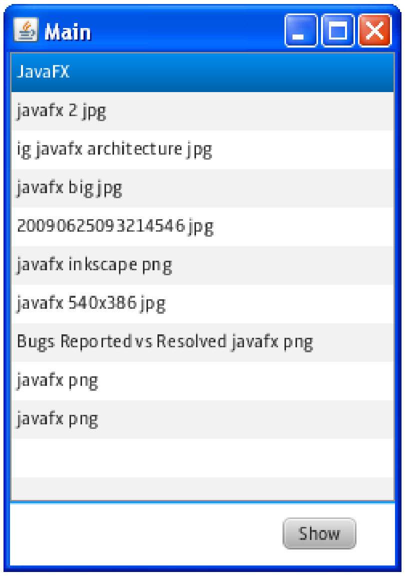


Select the "Design File | Resources | image [Image]" node in the Navigator and go to the Properties window. There change the value of the "Identifier" property to "placeHolderImage".

Test the Service

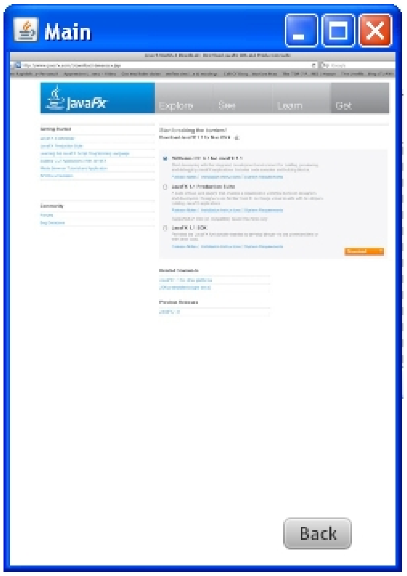
Run the application using the "Run Project" pop-up menu action on the "ImageViewer" project node in the Projects window.

The application is built and started. You should see an empty listing and after a few seconds (after the application connects and receives the image search results), the listing should show the list of images the application has found.



JavaFX
javafx 2 jpg
ig javafx architecture jpg
javafx big jpg
20090625093214546 jpg
javafx inkscape png
javafx 540x386 jpg
Bugs Reported vs Resolved javafx png
javafx png
javafx png

Select an image and press the "Show" button. The scene slides to the left and the "Loading..." image is replaced with the selected image.



Exercise 5: Data Source Customized

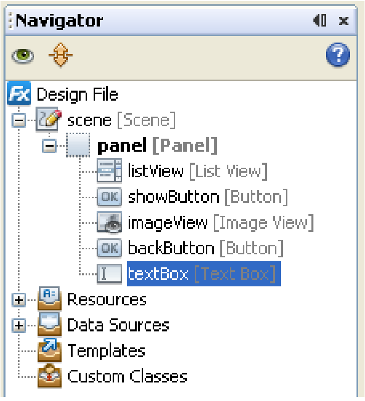
This is an optional exercise that enhances the "ImageViewer" application with a text-box to specify a search keyword and a progress-indicator to show that the data source is loading data.

Adding Search Box

Select the Master state in the States editor. Invoke the "Design Component" pop-up menu action on the "Design File | scene [Scene] | panel [Panel]" node in the Navigator window.

Select the "listView" component. Go to the Properties window. Set the value of the "Layout Y" property to "40" and the value of the "Height" property to "240". Or you can move and resize the "listView" directly in the Design editor. If you cannot find the "Layout Y" and "Height" properties, press the "Layout" button in the tool-bar of the Properties window.

Go to the Palette and drag and drop the "Text Box" item in the "Controls" category into the "panel" container. A new "textBox" component is added into the "panel" container. Check the structure in the Navigator.



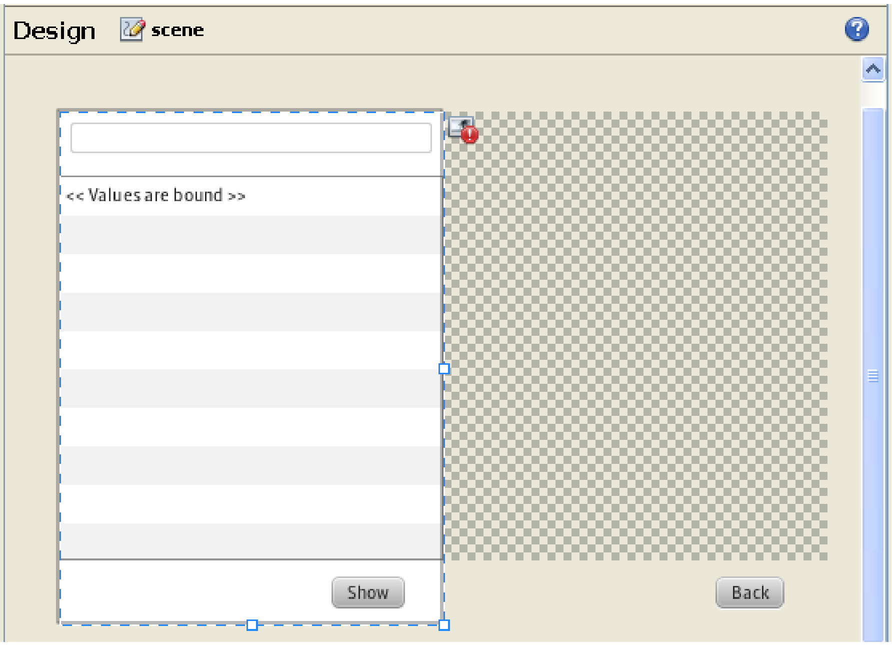
Navigator

Design File
- scene [Scene]
  - panel [Panel]
    - listView [List View]
    - showButton [Button]
    - imageView [Image View]
    - backButton [Button]
    - textBox [Text Box]
- Resources
- Data Sources
- Templates
- Custom Classes

Move the "textBox" component to be 6 pixels from the top-left corner of the "panel" container. Change the "Width" of the "textBox" component so it occupies the whole width of the "Listing" scene.

Check the "textBox" component property values in the Properties window. The "Layout X" and "Layout Y" properties should be set to "6" and the "Width" property should be set to "228".

Click on the "scene" hyperlink on the right-side of the "Design" label in the breadcrumb in the Design editor to return to designing the whole scene. Your design should look like this:



Finally we have to integrate the input from the "textBox" component into the URL set in the "httpDataSource".

Select the "textBox" component. Go to the Properties window. There is an "Action" property. The "Action" property represents a function that is invoked when an user presses the Enter key while the "textBox" component is focused in the application.

Press the "Generate" button ✎. A pop-up menu appears. In the pop-up menu, select the "Generate: Empty function" item. The design window immediately switches to the Source editor. Then a new function is created with an empty body:

```
function textBoxAction(): Void {
    // TODO
}
```

Change the function to look like this:

```
function textBoxAction(): Void {
    def searchFor = javafx.io.http.URLConverter{}.encodeString(textBox.text);
    httpdataSource.url =
"http://search.yahooapis.com/ImageSearchService/V1/imageSearch?
output=json&appid=PigYsoHV34FfXCVua8VFP0a.YWCQ1l6dq_tBPYGn0PjYEmZd9RHVU8RW8zN
WcY_UBIKO&query={searchFor}";
}
```

You can see that the function just resolves a new URL which uses the "textBox.text" value as the text to be searching for and then the URL is assigned to the "url" property "httpDataSource". When assignment is done, the "httpDataSource" automatically connects to the Web Service, parses its answer and supply the parsed data to your application.

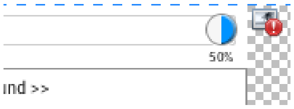
Press the "Design" button to switch the design window to Design editor.

Adding Progress Indicator

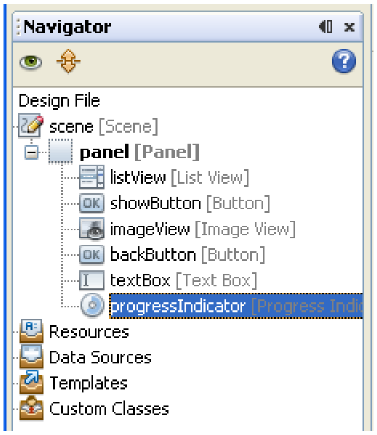
A Web Service or any other data source may have a delay in answering you. In this case, it is better to give users some feedback that the data-loading is in progress.

Select the Master state in the States editor. Invoke the "Design Component" pop-up menu action on the "Design File | scene [Scene] | panel [Panel]" node in the Navigator window.

Drag the "Progress Indicator" item in the Controls category in the Palette and drop it into the "panel" container somewhere on the right-size of the "textBox". For example, see the screenshot below, where it covers the right-most part of the "textBox" component:



The new "progressIndicator" component should be added into the "panel" container. Check the structure in the Navigator.



Design File
scene [Scene]
panel [Panel]
listView [List View]
showButton [Button]
imageView [Image View]
backButton [Button]
textBox [Text Box]
progressIndicator [Progress Indic
Resources
Data Sources
Templates
Custom Classes

Again return from designing the "panel" container by clicking on the "scene" hyperlink on the right-side of the "Design" label in the breadcrumb in the Design editor.

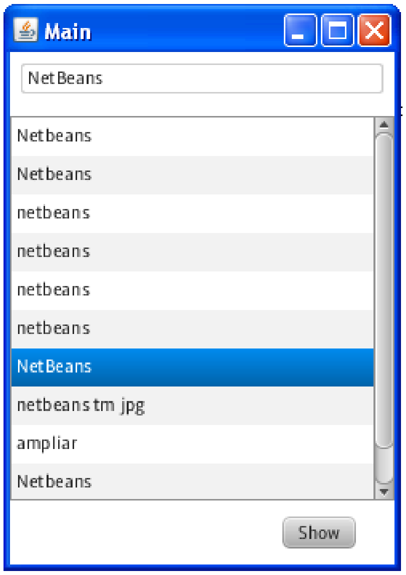
Select the "progressIndicator" component. Go to the Properties window. Press the "Details" button ▭ for the "Visible" property. The "Details" window appears. In the "Details" window, press the "Bind" button to switch to bind mode. The binding expression should be empty now. Select "httpdataSource["ResultSet/Result"]" in the left-most list and select "fetchingData" in the middle list. The following binding expression should be in the text-field:

```
httpdataSource.getDataSource("ResultSet/Result").fetchingData
```

Run the Application

The application is built and started. The application should start as before except that a progress indicator is shown while connecting and parsing an answer from the Web Service. When the progress indicator disappears, the listing should be filled with the images as before.

Focus the text-box at the top of the screen. Type "NetBeans" into it and press the Enter key. The progress-indicator should appear again and the listing should be refreshed with the search results.



NetBeans

Netbeans
Netbeans
netbeans
netbeans
netbeans
netbeans
NetBeans
netbeans tm jpg
ampliar
Netbeans

Exercise 6: Multiple State Variables

This is an optional exercise that enhances the "ImageViewer" application with a form with more information about image that the application has found.

Introduction

In a typical application, you do not have just a single state variable. You can create another state variable with another sequence of states. These states independently control another set of properties in your design.

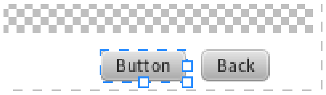
NetBeans JavaFX Composer allows you to compose your application not only using pre-defined components available in JavaFX 1.3 SDK but so-called templates too. The templates are more complex structures that consists of a hierarchy of standard components. They can be easily customized to change their structure.

Adding the More Button

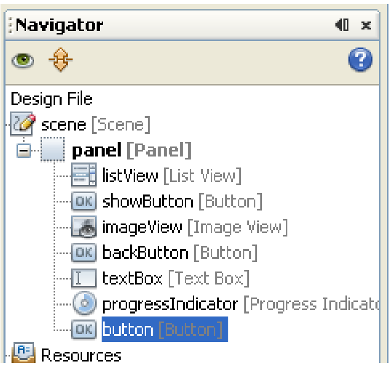
Select the Master state.

Invoke the "Design Component" pop-up menu action on the "Design File | scene [Scene] | panel [Panel]" node in the Navigator window.

Drag the "Button" item in the Controls category in the Palette and drop it into the "panel" somewhere on the left-side of the "detail" part of the scene, as it looks here:



The new "button" component should be added into the "panel" container. Check the structure in the Navigator.

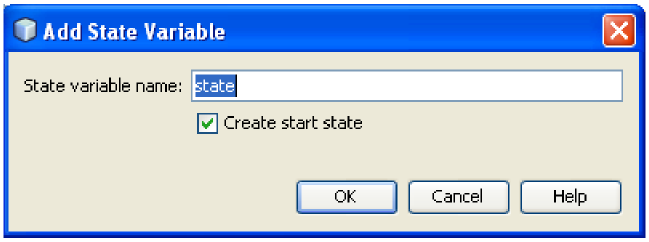


You need to edit the "button" component. Select it in the Design editor. Go to the Properties window. Change the value of "Identifier" property to "moreButton". Change the value of the "Text" property to "More".

Adding State Variable

Now we are going to add another state variable that will control the appearance of the "form".

Press the "Add" button in the tool-bar in the State Variables editor. A dialog appears:

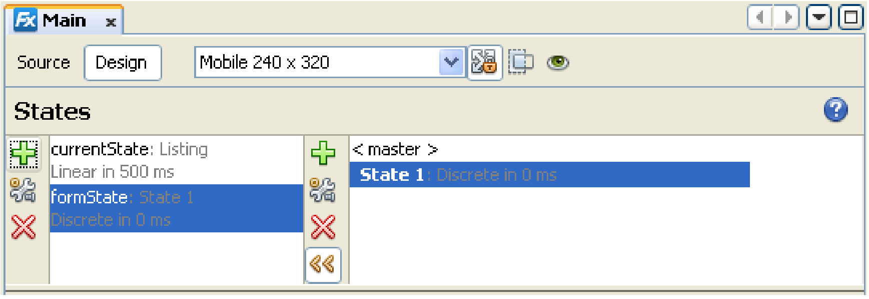


Add State Variable

State variable name: state

☑ Create start state

In the dialog, set the value of the "State variable name" text-field to "formState". Press OK to close the dialog. The "formState" state variable is added to the list and selected. See that the States editor now contains the Master state and one pre-created derived state called "State 1".

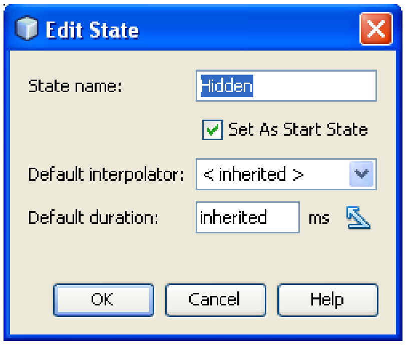


currentState: Listing
Linear in 500 ms
formState: State 1
Discrete in 0 ms

< master >
State 1: Discrete in 0 ms

Select the "State 1" state and press the "Edit" button in the tool-bar in the States editor.

In the dialog, change the text in the "State name" text-field to "Hidden":



Edit State

State name: Hidden

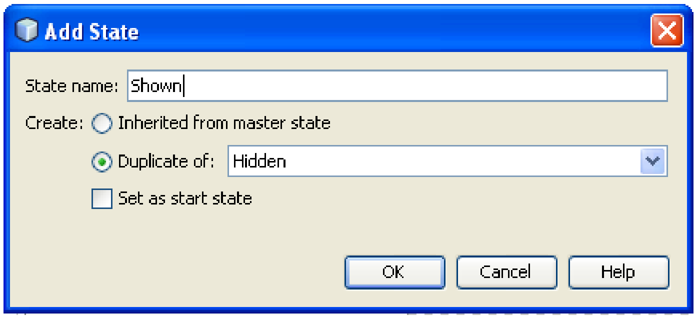
☑ Set As Start State

Default interpolator: < inherited >

Default duration: inherited ms

Press "OK" to close the dialog.

Press the "Add" button in the tool-bar in the States editor. In the dialog, change the text in the "State name" text-field to "Shown":



State name: Shown

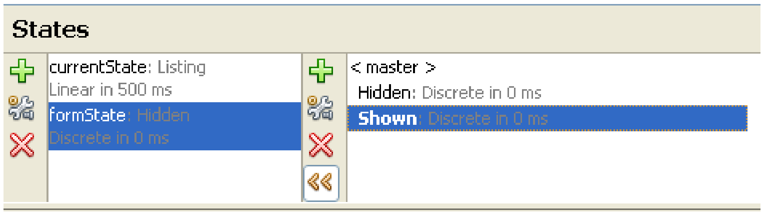
Create: ○ Inherited from master state

● Duplicate of: Hidden

□ Set as start state

Press "OK" to close the dialog.

Now the "formState" states should look like this:



currentState: Listing
Linear in 500 ms

formState: Hidden
Discrete in 0 ms

< master >

Hidden: Discrete in 0 ms

Shown: Discrete in 0 ms

Editing Form States

Keep the "formState" state variable selected in the state variable editor. So far we have not modified the design in either the "Hidden" or the "Shown" state so they both look the same.

In the Master state we want to do just one thing - define the "Action" of the "moreButton" component.

Select the Master state in the states editor. Select the "moreButton" component and go to the Properties window. Press "Generate" ✎. In the pop-up menu, select the "Generate: Go to next state" item. The design window immediately switches to the Source editor. Then a new function is created with a pre-filled body:

```
function moreButtonAction(): Void {
    currentState.next();
}
```

Change the function to look like this:

```
function moreButtonAction(): Void {
    formState.nextWrapped();
}
```

Note that we are using "formState" instead of the "currentState" state variable that was pre-filled in the function body.

Note that the "nextWrapped" function replaces the "next" function. The "nextWrapped" function automatically cycles through the sequence of states from the beginning in case it reaches its end.

Press the "Design" button to switch the design window to the Design editor.

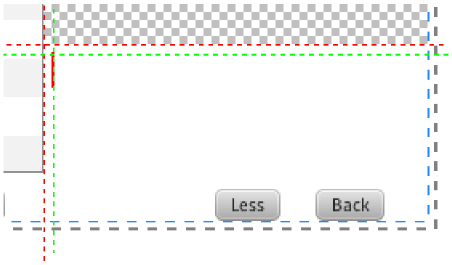
Do not modify anything in the "Hidden" state since we want to keep it exactly as it is defined in the Master state.

We want to modify the design in the "Shown" state. Select the "Shown" state in the states editor.

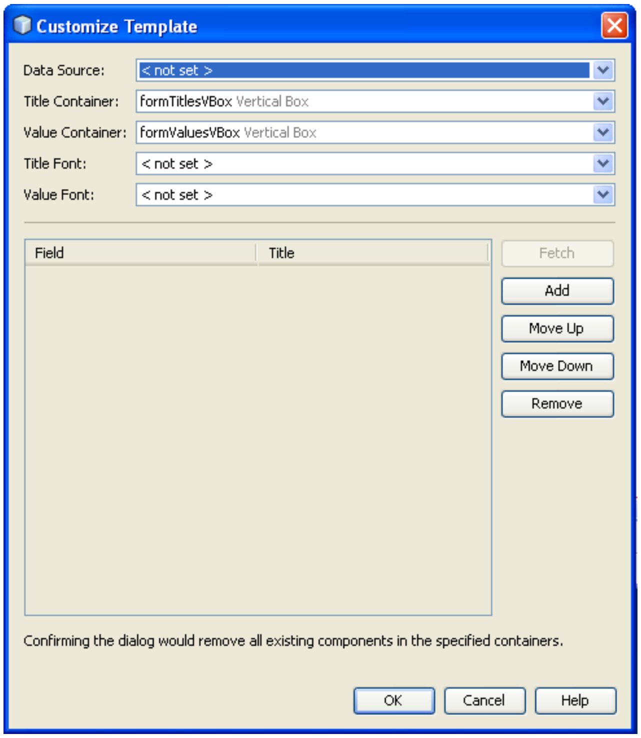
Select the "moreButton" component in the Design editor. Go to the Properties window. Set the value of the "Text" property to the "Less" text.

Select the "imageView" component in the Design editor. In the Properties window, set the value of the "Fit Height" property to "200". Or you can resize the height in the design editor directly.

Keep the "Shown" state selected. Go to the Palette. Scroll to the bottom. Drag the "Desktop Form" item from the "Templates" category in the Palette to the Design editor right below the "imageView" component.



When you drop the item to the Design editor, a Form customizer dialog appears:


Customize Template

Data Source: < not set >
Title Container: formTitlesVBox Vertical Box
Value Container: formValuesVBox Vertical Box
Title Font: < not set >
Value Font: < not set >

Field	Title

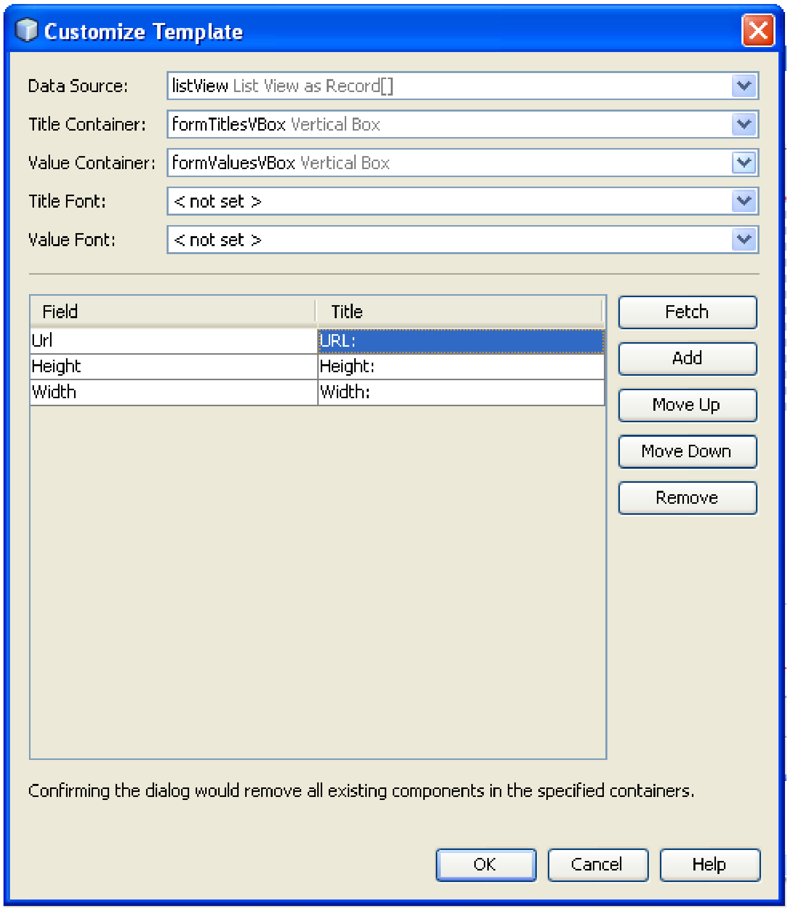
Confirming the dialog would remove all existing components in the specified containers.

The dialog allows you to customize the form. The form template creates a form-like structure of Labels. These Labels represent titles and values of attributes of the Record that is active on the selected data source.

Select the "listView List View as Record[]" item in the "Data Source" combo-box. This way the active Record is taken from "listView.selectedItem" property. Press the "Fetch" button. Now the table is filled with all attributes that are known to be in a Record in the "listView" component. Select all fields except the following ones and press "Remove" button:

- Url - The URL of the found image
- Height - The height of the found image
- Width - The width of the found image

Double-click on the cell for the "Title" column of the "url" attribute row. Enter the text "URL:" instead and press the Enter key to close the in-place editor. The customizer should look like this:



Customize Template

Data Source: listView List View as Record[]

Title Container: formTitlesVBox Vertical Box

Value Container: formValuesVBox Vertical Box

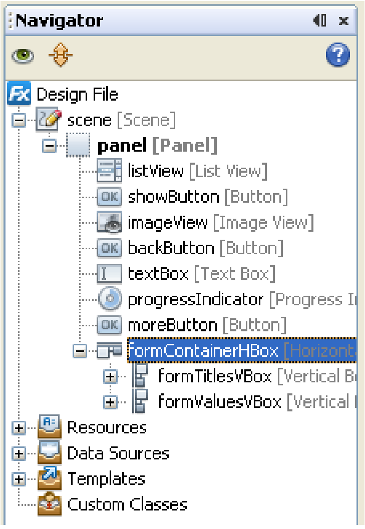
Title Font: < not set >

Value Font: < not set >

Field	Title
Url	URL:
Height	Height:
Width	Width:

Confirming the dialog would remove all existing components in the specified containers.

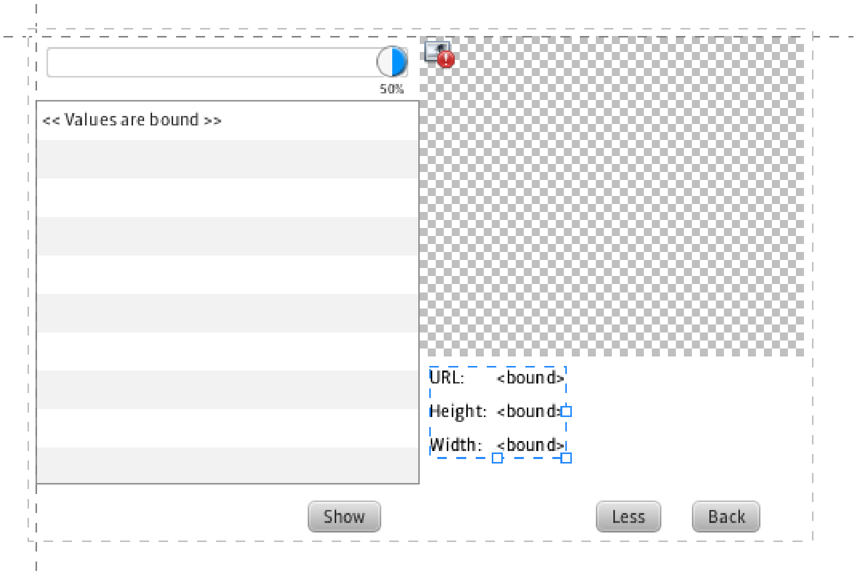
Press "OK" to close the dialog. Now the form structure is created in your design. It should be placed into the "panel" container. Check it in the Navigator window:



Navigator

Design File
- scene [Scene]
  - panel [Panel]
    - listView [List View]
    - showButton [Button]
    - imageView [Image View]
    - backButton [Button]
    - textBox [Text Box]
    - progressIndicator [Progress I
    - moreButton [Button]
    - formContainerHBox [Horizont
      - formTitlesVBox [Vertical B
      - formValuesVBox [Vertical
- Resources
- Data Sources
- Templates
- Custom Classes

In the Design editor, the form structure should look like this:



You may select and move the "formContainerHBox" container in the Design editor to fine-tune its location.

Note that the we have added the form template into our design while the "Shown" state has been selected. Therefore the form structure ("formContainerHBox") is visible in the "Shown" state only and invisible everywhere else. You can check it by selecting Master and other states.

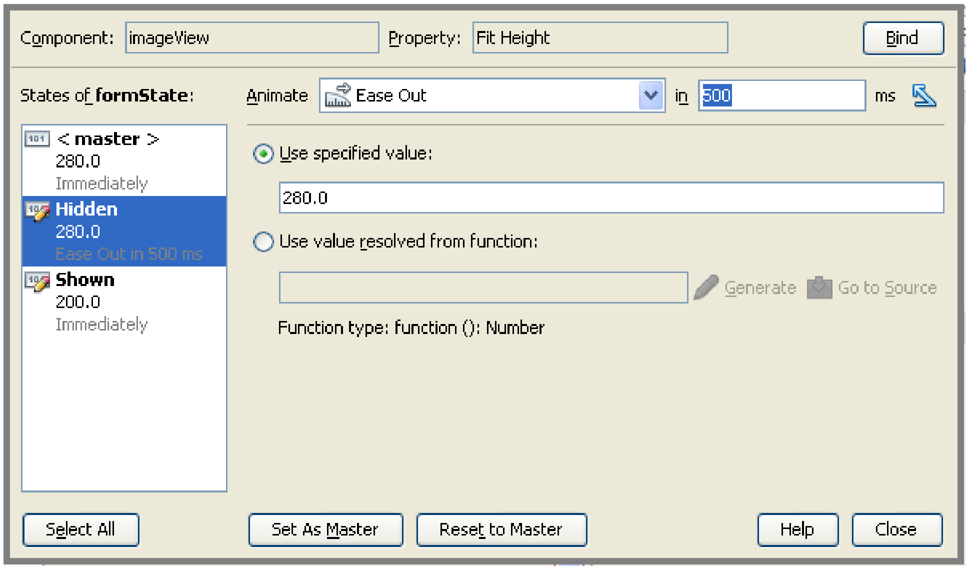
Click on the "scene" hyperlink on the right-side of the "Design" label in the breadcrumb in the Design editor to return back to designing the whole scene.

Animating Image View

We may improve our design with an animation of growing "imageView" component in case the information form gets hidden.

Select the "Hidden" state in the states editor. Select the "imageView" component in the Design editor. Go to the Properties window. Press the "Details" button for the "Fit Height" property to open the "Details" window.

In the window, edit the animation in the top-right corner. Set "Ease Out" item in the interpolation combo-box and enter "500" in the duration text-field. Note that you have to have the "Hidden" state selected in the states list on the left-side of the window:



Component: imageView
Property: Fit Height
States of **formState**:
Animate: Ease Out in 500 ms

< **master** >
280.0
Immediately

Hidden
280.0
Ease Out in 500 ms

Shown
200.0
Immediately

● Use specified value: 280.0
○ Use value resolved from function:
Function type: function (): Number

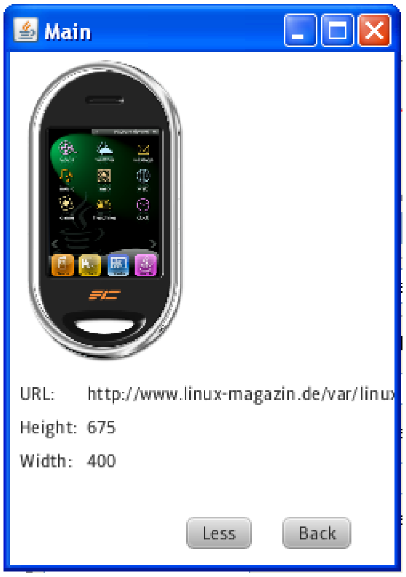
Press "Close" to close the window.

Running the Application

Run the application using the "Run Project" pop-up menu action on the "ImageViewer" project node in the Projects window.

The application is built and started. You can select an image in your list. Press "Show" to slide to the "Detail" state. You should see an image shown across the whole scene except for the Button area.

Press the "More" button. More information should be shown for the currently shown image:

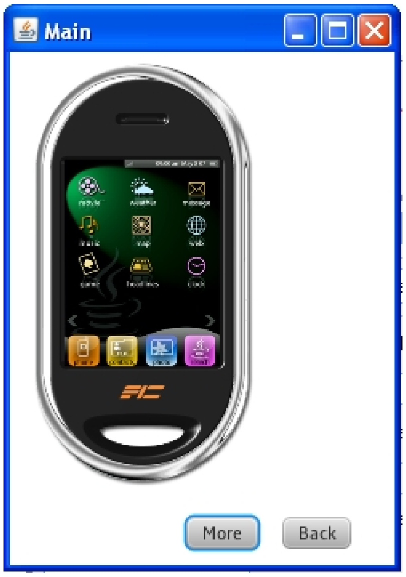


URL: http://www.linux-magazin.de/var/linux

Height: 675

Width: 400

Press the "Less" button and the information gets hidden and the image is smoothly enlarged to fill the scene again:



Summary

Congratulations! You have successfully completed LAB S313803: NetBeans JavaFX Composer in Action – Creating Applications with Rich UI. In this lab you learned about:

- creating a dynamic UI using concept of states and
- connecting your application to data from a web-services using concept of data sources.

For additional information about the technologies and the tool used in this lab, please set the following links:

- http://www.netbeans.org/kb/trails/matisse.html - tutorials
- http://wiki.netbeans.org/JavaFXComposer - the home page of the project – contains the latest update about the releases, features, tutorials
- http://www.netbeans.org/downloads/ - the download page

If you have questions or comments about the lab, you can post them on the public product forum here:

- http://forums.netbeans.org/netbeans-users.html – the NetBeans Users forum

Thank you for participating!

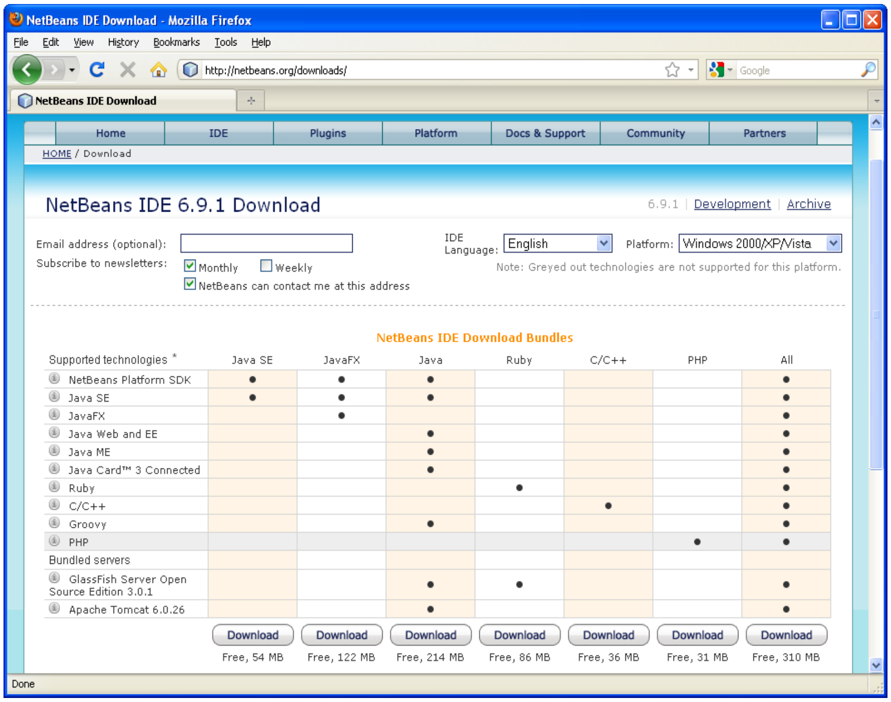
Performing This Lab Outside of the JavaOne Environment

Requirements

For this HOL, you need a Windows XP machine with JDK 6 Update 20 installed.

Installation

For this HOL, you need to have NetBeans 6.9.1 with JavaFX support installed. It can be downloaded at http://www.netbeans.org/downloads/. Click on the "Download" button at the "JavaFX" column. Make sure that you have the "6.9.1" release selected at the right-top corner of the web-page.


NetBeans IDE Download - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://netbeans.org/downloads/
Google
NetBeans IDE Download
Home IDE Plugins Platform Docs & Support Community Partners
HOME / Download
6.9.1 | Development | Archive
Done

NetBeans IDE 6.9.1 Download

Email address (optional):

Subscribe to newsletters: ☑ Monthly □ Weekly

☑ NetBeans can contact me at this address

IDE Language: English

Platform: Windows 2000/XP/Vista

Note: Greyed out technologies are not supported for this platform.

NetBeans IDE Download Bundles

Supported technologies *	Java SE	JavaFX	Java	Ruby	C/C++	PHP	All
NetBeans Platform SDK	●	●	●				●
Java SE	●	●	●				●
JavaFX		●					●
Java Web and EE			●				●
Java ME			●				●
Java Card™ 3 Connected			●				●
Ruby				●			●
C/C++					●		●
Groovy			●				●
PHP						●	●
Bundled servers							
GlassFish Server Open Source Edition 3.0.1			●	●			●
Apache Tomcat 6.0.26			●				●
	Download	Download	Download	Download	Download	Download	Download
	Free, 54 MB	Free, 122 MB	Free, 214 MB	Free, 86 MB	Free, 36 MB	Free, 31 MB	Free, 310 MB

After you download the installer, install the IDE and run it.