WaveMaker Studio Tutorial

VERSION 3.2

WaveMaker

WaveMaker Copyright

Notices

WaveMaker Studio Tutorial

This tutorial walks you through the steps for creating an application in WaveMaker Visual AJAX Studio™. The simple application that you build here will include AJAX widgets, a database and a Web Service. The application is broken down into three separate tutorials, the first of which should take about 30 minutes to complete:

1. "Create a New WaveMaker Studio Application" on page 4 [30 minutes]
2. "Add a Customer Detail Tab" on page 25 [20 minutes]
3. "Add a Web Service Call" on page 38 [25 minutes]

Before you begin these tutorials, review the requirements in the following section ("Before You Begin" on page 1).

Before You Begin

Before you begin, make sure you:

- "Install WaveMaker Studio" on page 1
- "WaveMaker Studio Orientation" on page 1

Install WaveMaker Studio

Run the WaveMaker Studio installer. This installs WaveMaker Studio, Java, Tomcat (Apache Tomcat 5.5.23) and Hypersonic SQL DB (HSQLDB). Tomcat and JDK are required by WaveMaker Studio. HSQLDB is an embedded database used by this tutorial, as well as many of the sample applications.

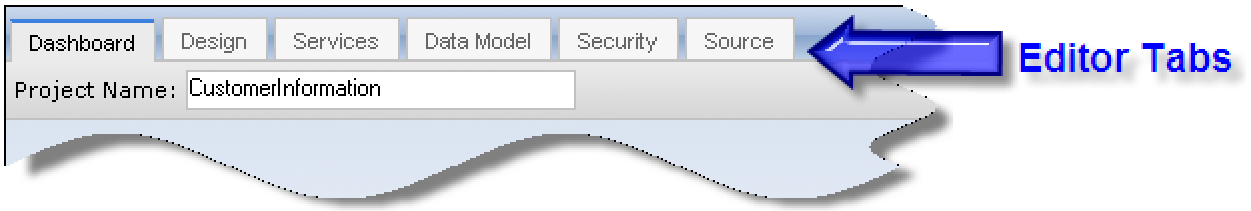
NOTE: The installer does not overwrite existing Java or Tomcat installations. If you already have Java or Tomcat installed, then the WaveMaker Studio creates new installations.

WaveMaker Studio Orientation

This section provides a quick orientation to the major elements of the WaveMaker Studio product. WaveMaker Studio is a web application that runs in your browser. Firefox is the recommended browser for creating your applications with WaveMaker Studio. Internet Explorer (IE) 7.0 is also supported. On earlier versions of IE, you might experience problems.

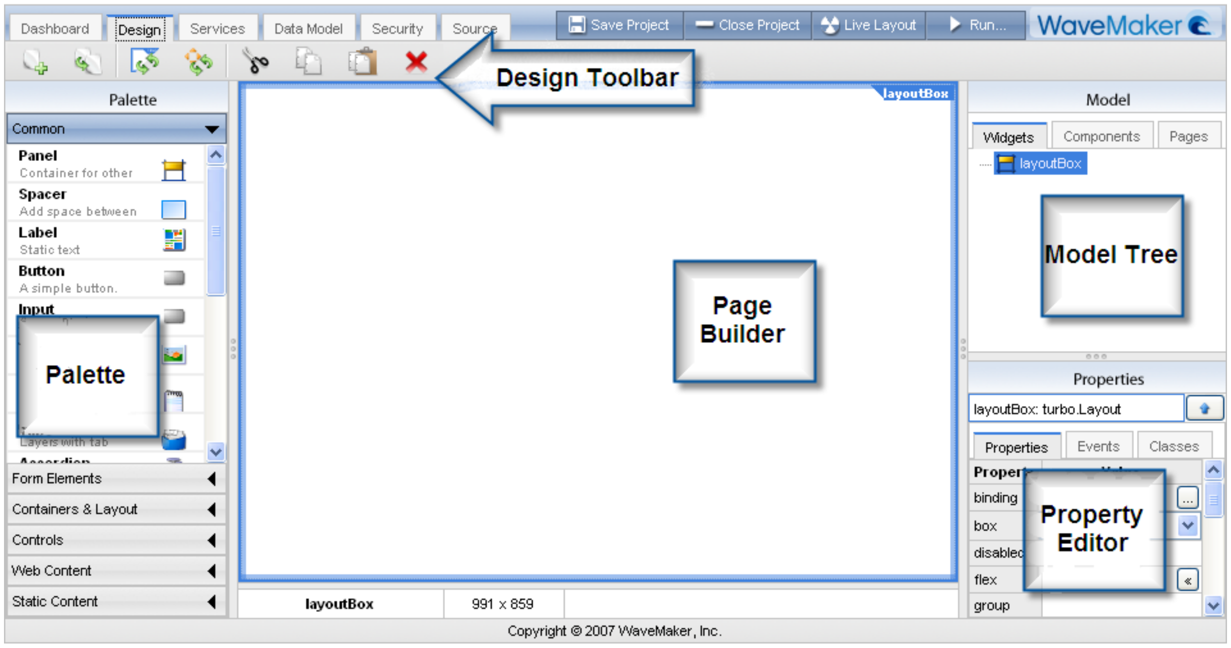
NOTE: The restrictions on browsers pertain to development time only. Once you have created your application, you should be able to run it on most browsers.

WaveMaker Studio includes a variety of Editors that allow you to import databases or web services, set up security, edit underlying code, and so on. You can access these Editors from the Editor Tabs that are always visible across the top left of the Studio.



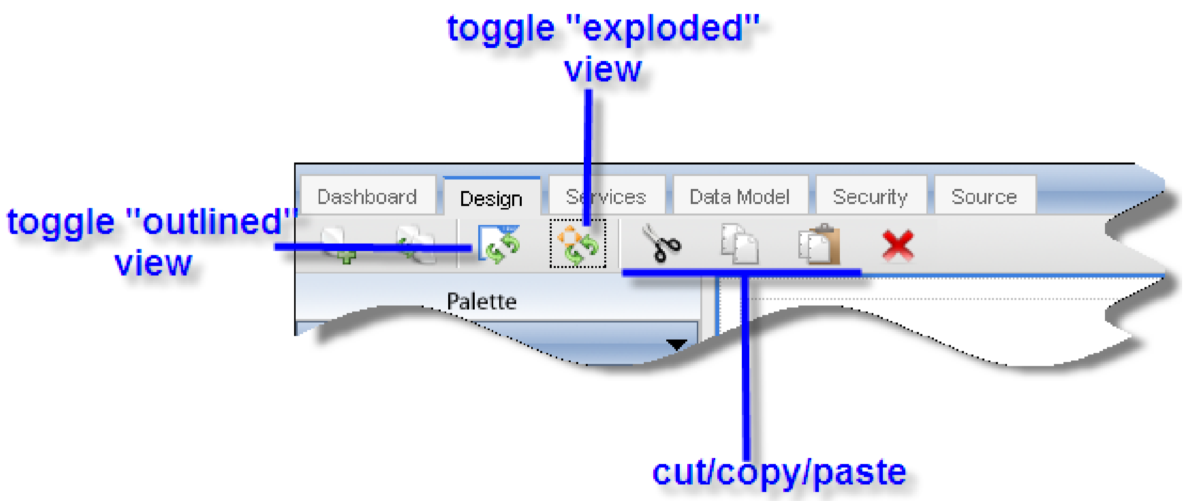
Project Name: CustomerInformation

The most important editor is the Page Designer, which you use to build your application pages. You can open the Page Designer from the Editor Tabs (click the **Design** tab). To build a page, you drag widgets from the Palette onto the Page and then configure them by editing their properties in the Property Editor. The following illustration shows the main parts of the Page Designer.



- **Page Builder:** The large area in the center of the Page Designer represents the page itself. Drag widgets from the Palette onto the page. (The whole page is contained in a container called the LayoutBox widget. It cannot be removed.)
- **Palette:** The Palette is located on the left side of the Page Designer. The Palette allows you to add content to your application, such as widgets, gadgets and sub-pages. In the Palette, select a category to expand the list of items in that category.

  The top category, called “Common” contains commonly-used page widgets. These common page widgets are also included in their logical groups, such as “Form Elements” or “Controls”.
- **Design Toolbar:** This toolbar provides buttons for common Page Designer functions. A few useful functions to note:
  - The cut and paste buttons allow you to cut a widget out of one Panel and paste it into another.
  - The exploded view puts a bit of padding between all the widgets so that they are easy to select. You can toggle it off and on with the exploded view button in the Page Designer toolbar. The exploded view can obscure widget alignment, so use it only when you need it.
  - The “outline” toggle button allows you to see your page without the widget outlines (the run-time look, rather than the design-time look). You typically work with the outlines on, then toggle them off to see what the page would look like at run time.



- **Model Tree:** The Model Tree is located on the upper right side of the Page Designer. The Model Tree has three tabs: a **Widgets** tab, a **Components** tab and a **Pages** tab:
  - The **Widgets** tab shows a hierarchical tree of all the widgets in the current page. When you select a widget in this tree, that widget is also selected in the Page Builder.

  - The **Components** tab shows the Service Calls, Navigation components, and Variables available to your page. Click on an object to see that object's properties in the Property Editor. You can also click on the Variable icon to create a new Variable or the Service Call icon to create a new Service Call.
  - The **Pages** tab shows all the pages in the current project. Double-click on a page in the list to open that page in the Page Designer.
- **Property Editor:** The Property Editor is located on the lower right of the the Page Designer. The Property Editor has three tabs: a **Properties** tab, an **Events** tab, and a **Styles** tab.

Create a New WaveMaker Studio Application

This tutorial walks you through the steps for creating a simple WaveMaker Studio application:

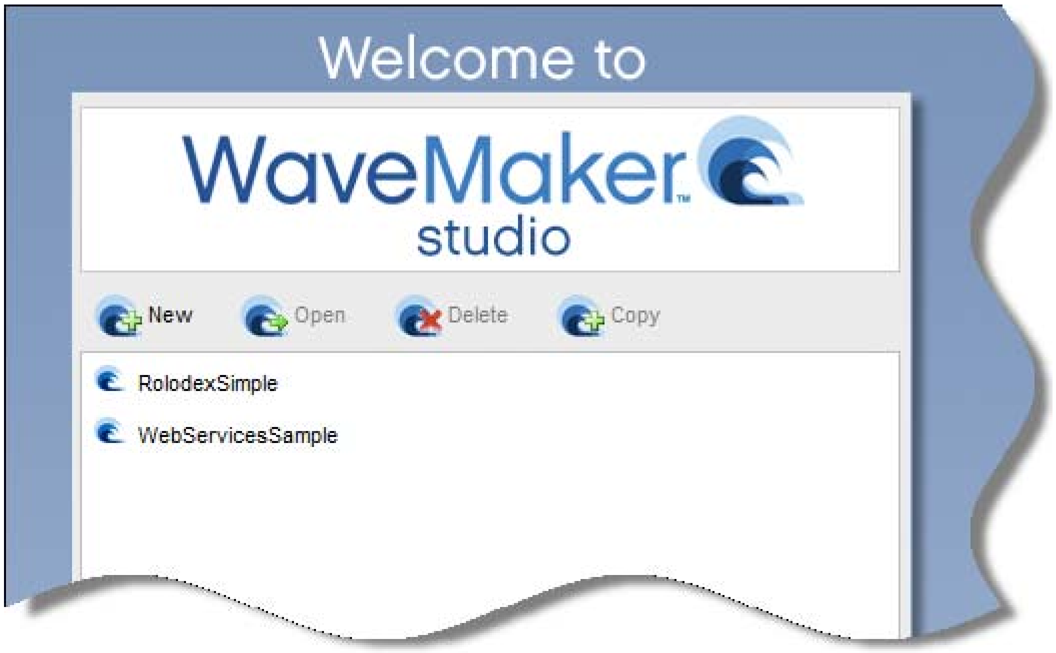
1. "Create the Project" on page 4
2. "Define the Main Application Page" on page 5
3. "Import the Database" on page 14
4. "Add Services to Call the Database" on page 17
5. "Configure the Controls" on page 20
6. "Run the Project" on page 24

Create the Project

To build a WaveMaker Studio application, you create a *project*. A project is the collection of all the files and references for the application. The first step to building our example application is to create a new WaveMaker Studio project:

1. Run WaveMaker Studio. (On Windows, choose **Start > All Programs > WaveMaker > WaveMaker**. On Linux, run the **wavemaker.sh** script.)

   WaveMaker runs in your browser. The first screen is always the WaveMaker Studio "Welcome" screen. From here you can open an existing project, copy a project or create a new one. There are some sample applications included with the installation.



2. Click **New Project**. The New Project Name dialog box appears. Type in a project name. Your project name can use only alphanumeric characters, without spaces. Type in:

   CustomerInformation

3. Click **OK**. WaveMaker Studio creates your new project including a single default page, called "Main". WaveMaker Studio opens this default page for you in the Page Designer.

At this point we can choose either to set up the data model or to lay out the main application page. WaveMaker Studio does not require you to set up the data model before you lay out the pages. In this example, we'll lay out the "Main" page first ("Define the Main Application Page" on page 5) and then set up the data model ("Import the Database" on page 14).

Define the Main Application Page

In this section, we set up our "Main" application page. There's nothing special about the "Main" page. It is simple an empty page that WaveMaker Studio generated for us when we created the project. By default, it is set up as the page that initially displays when the application is run.

This tutorial uses the "Main" page, but you do not have to. Here are the steps to set up the page:

- "Set up the Initial Layout" on page 6
- "Add Header and Footer Markup" on page 8
- "Set up the Search Panel" on page 10

- "Add the DataGrid" on page 14

Set up the Initial Layout

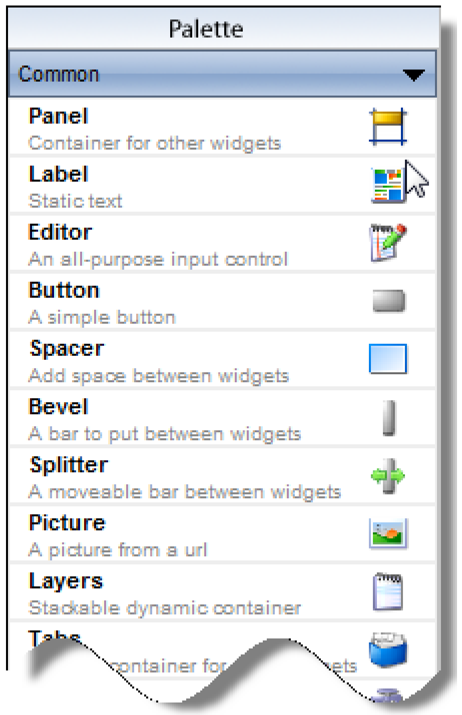
The project that we're going to build in this example begins with a single search/list page. We have a database that contains customer/order information. We want to create a simple page that allows users to search for a customer by company name and display a list of companies that match the search characters. Later we'll add more functionality to this basic application.

We will use the automatically-generated project page, called "Main" to lay out our content. We want a header and a footer and a "layers" section in the middle. The layers section holds our search/list widgets.

To set up the Main page, follow these steps:

1. If you don't already have the Main page open in the Page Designer, click the **Design** tab now to open it. The Main page contains a single widget, called "LayoutBox". The LayoutBox widget acts as the top container for each page. Each page has one (and only one) LayoutBox widget. You can't remove it and you can't put anything outside of it.
2. Add the header, footer and container widgets to the LayoutBox widget. To add a widget, drag it from the Palette and drop it where you want it to go.

   NOTE: The Palette has different groups of widgets, such as "Form Elements", "Controls", "Web Content' and so on. All the most commonly-used widgets from all these groups are also available in the "Common" section of the Palette. In this example, we drag all our widgets from the "Common" section.


Palette
Common
Panel
Container for other widgets
Label
Static text
Editor
An all-purpose input control
Button
A simple button
Spacer
Add space between widgets
Bevel
A bar to put between widgets
Splitter
A moveable bar between widgets
Picture
A picture from a url
Layers
Stackable dynamic container

From the “Common” Palette, drag the following widgets into the LayoutBox widget:

- Two *Content* widgets. A “Content” widget is a widget that you can define by pasting in your own markup. These widgets are especially useful for adding your corporate look-and-feel to headers and footers.
- A *Layers* widget. The “Layers” widget defines an area where different content can be displayed. For the initial search/list part of the application we don’t really need to use a Layers widget—we will need it later when we add a tab to show customer details.

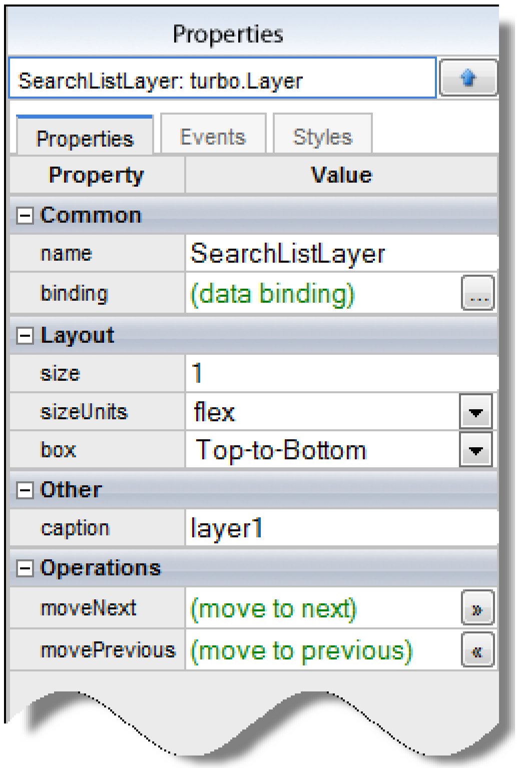
Layers: With Web 2.0 AJAX-enabled applications, the only thing the client typically gets from the server is the data. When you build a Web 2.0 application with WaveMaker Studio, you might build only one “page”. That single page contains multiple *layers* that can be dynamically revealed or hidden. To show a different layer, you don’t need to get it from the server because the client already has it. You do not destroy the existing context.

To use the Layers widget, we add a layer for each “page” of content we plan to display. For this application we’ll ultimately have two tabbed layers: one for customer search and list and one to add a new customer. For now, we just need the first layer, which is automatically added.

3. Drag the Layers widget between the two Content widgets so that the first Content widget is on top, the Layers widget is directly below it, and the second Content widget is on the bottom.

4. When you add the Layers widget, a single layer is automatically added with it. In the Model Tree, select the new layer (the default name is “layer1”).
5. Each widget has a **name** property that uniquely identifies it in the page. This is the name that you will see in the “Widgets” section of the Model Tree, for example. With the new layer selected, go to the Property Editor, located in the lower right corner of the WaveMaker Studio.

   The Property Editor lists all the properties that you can use to configure the selected widget. The Properties are organized into groups, such as “Common”, “Layout”, “Operations” and so on.
6. In the list of “Common” properties, for the **name** property type in: `SearchListLayer`



Property	Value
Common	
name	SearchListLayer
binding	(data binding)
Layout	
size	1
sizeUnits	flex
box	Top-to-Bottom
Other	
caption	layer1
Operations	
moveNext	(move to next)
movePrevious	(move to previous)

7. Save the project.

The next step is to paste in our header and footer markup (“Add Header and Footer Markup” on page 8).

Add Header and Footer Markup

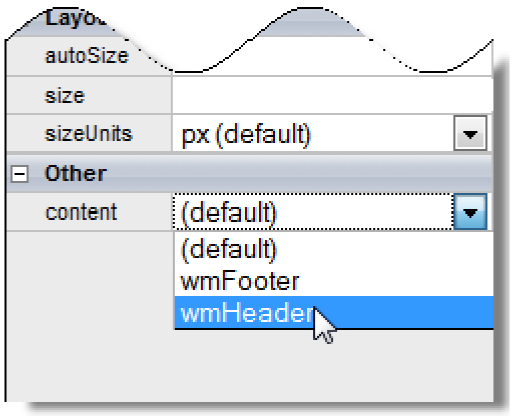
The reason we used *Content* widgets for the header and footer is that Content widgets allow you to easily use your own markup. The WaveMaker Studio installation includes the markup we use for these examples. Find it under the WaveMaker installation directory under **Support\Tutorial\css**. The name of the file is **WaveMakerCSSHTML.txt**.

For example, if WaveMaker is installed in the **C:\Program Files** directory, you would find the markup file here:

C:\Program Files\WaveMaker\Support\Tutorial\css\WaveMakerCSSHTML.txt

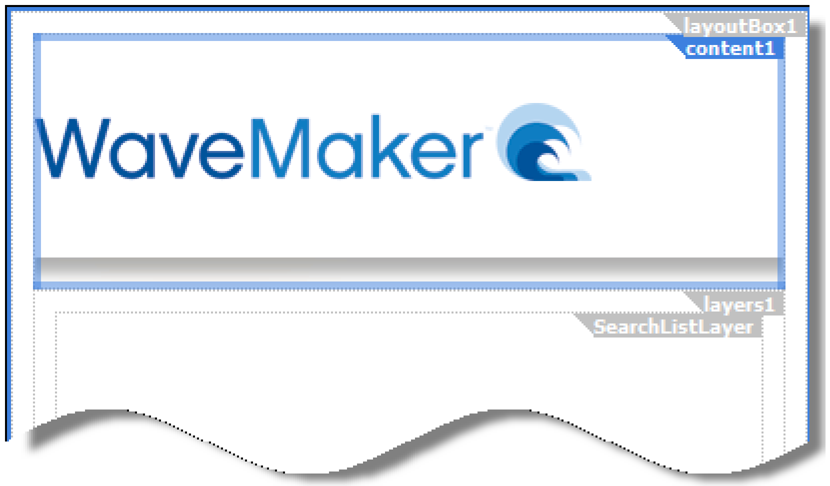
To add the markup to the Content widgets, follow these steps:

1. Click the **Source** tab along the top of the Page Designer. Several sub-tabs appear.
2. Click the tab labeled **Markup**.
3. Now open the **WaveMakerCSSHTML.txt** file in any text editor. Select and copy the contents of this file.
4. Paste the text from the **WaveMakerCSSHTML.txt** file into the text area under the **Markup** tab in the WaveMaker Studio. Notice that this markup contains an id called *wmHeader* and one called *wmFooter*.
5. Save the project.
6. Click on the **Design** tab to bring up the Page Designer again.
7. Select the first Content widget (the one on the top of your page).
8. In the Property Editor, under the "Other" section, find the property called **content**. The drop-down menu for this property now contains the header and footer markup we just added. Select **wmHeader** from the drop-down menu.



When you select **wmHeader** for the **content** property, your custom header appears in the Content widget. The header probably doesn't fit in the Content widget, so we will use the **autoSize** property to make the Content widget stretch or shrink to fit the content.

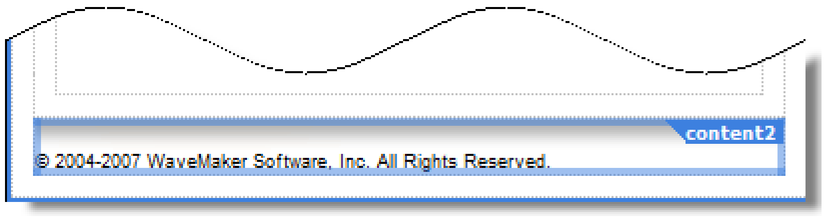
9. In the Property Editor, under the "Layout" section, click to check the **autoSize** checkbox. This sizes the widget to fit the content.



10.Select the second Content widget.

11.In the Property Editor, select **wmFooter** from the drop-down menu for the **content** property.

12.In the Property Editor, under the "Layout" section, click to check the **autoSize** checkbox. This sizes the Content widget to fit the footer content.



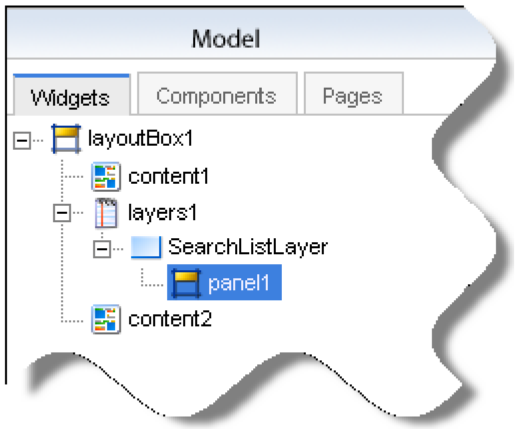
13.Save the project.

The next step is to set up the search Panel ("Set up the Search Panel" on page 10).

Set up the Search Panel

Now we'll add the Search form to the application:

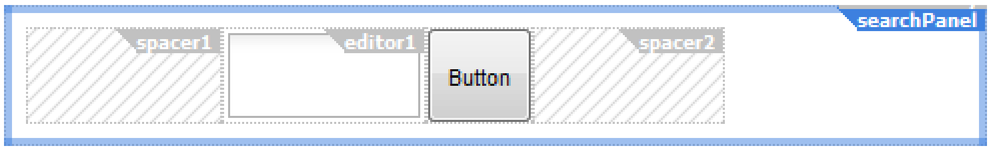
1. Drag a Panel widget into the SearchListLayer. Panel widgets are simple container widgets that hold other widgets on the page. When you have the Panel widget on the page, the Model Tree should look something like this:



2. Leave the Panel widget selected in the Model Tree. In the Property Editor, change the name of the **name** property from "panel1" to "searchPanel". This makes it easier to identify this Panel in the Model Tree.

3. Now we add the search input and button, along with spacers to center our search form on the page. From the "Common" Palette, drag the following widgets into the Panel widget you just created (searchPanel):
   - ∞ a *Spacer* widget
   - ∞ an *Editor* widget
   - ∞ a *Button* widget
   - ∞ another *Spacer* widget

4. With both the outlined and exploded views turned on, your panel widget should look something like this:



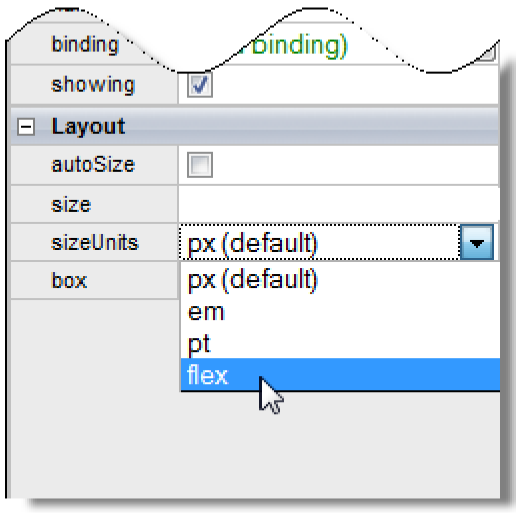
5. Drag the widgets inside the panel until they are in the order shown above: Spacer, Editor, Button, Spacer.

6. We want to space the widgets appropriately within the Panel. The Panel arranges its contents horizontally. To change the height of the widgets inside, we would need to change the height of the Panel itself. However, we can size each individual widget *horizontally* within the Panel.

Note: This Panel arranges its contents horizontally because its **box** property is set to **Left-to-Right**. This means that all the widgets we add to the Panel are displayed horizontally, side by side. If we were to change the value of the **box** property to **Top-To-Bottom**, then all the widgets inside the Panel would be arranged vertically.

7. Select the Editor widget and move your cursor over the right edge until you see the resize arrows. Make the Editor widget wide enough for a label and an input.

8. We are going to use a special sizing unit called "flex" to size our Spacer widgets. Click to select the first Spacer widget. In the Property Editor, under the "Layout" section, choose **flex** from the **sizeUnits** drop-down menu.

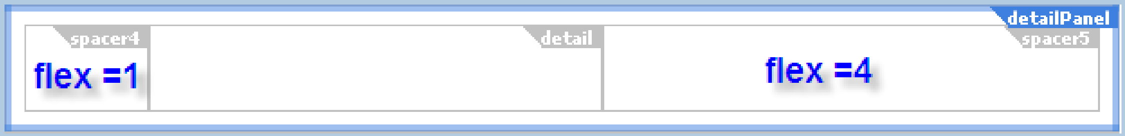


9. For the **size** property, the value is automatically changed to:

   1

10. Leave that value. Repeat this process for the second spacer widget. Both widgets should have the **size** property set to "1" with the **sizeUnits** property set to "flex".

Flex: Flex sizing is very useful for arranging content on a page. If you set a widget's flex size to 1, then that widget will fill all the remaining space on page. If two or more widgets in the same container have a flex, then the flex sizes work as a ratio. For example, if one widget has a flex of 1 and another a flex of 4, the second one will be four times bigger than the first.



Keyboard Shortcut: Select a widget, use <Ctrl> f to set sizeUnits to "flex" and value to "1".

11. The Editor widget provides both a label and a text input for the search field. The Editor widget has properties that allow us to configure both the caption (label) and the input field. Select the Editor widget.

12. In the Property Editor, under the "Display" section, we have a variety of properties for configuring the caption. The caption is the label that the users see displayed for the input field. (WaveMaker uses the **caption** property to hold labels that *the user* sees; it uses the **name** property to identify the widget for *the developer* within WaveMaker Studio.)

    Set the following properties for the Editor caption:

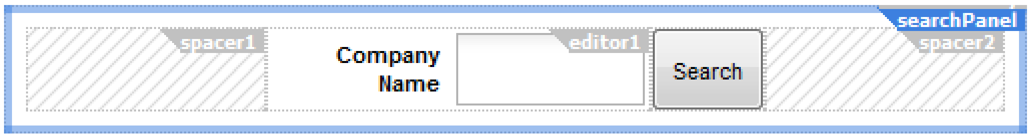
    - For the **caption** property, type in:
      `Company Name:`
    - For the **captionUnits** property, select **px** from the drop-down menu. This means that the **captionSize** will be measured in pixels.
    - For the **captionSize** property, type in:
      `80`

13. Finally, we want to change the text on the Button widget. Select the Button widget.

14. In the Property Editor, under the "Common" section, for the **caption** property, type in:
    `Search`

15. Next, we change the height of the Panel so that the button and input look good. Select the Panel and then move your cursor over the bottom of the Panel until you see the resize arrow. Resize the Panel so that is about the right height for a button and a text input.

    The Search Panel should look something like this:



16. Save the project.

The Search Panel is now complete, except that the button doesn't do anything yet. We'll handle that later. The next step is to set up the DataGrid widget to display our data ("Add the DataGrid" on page 14).

Add the DataGrid

Next we're going to add a DataGrid widget. A DataGrid widget is a widget especially designed to display lists of data.

1. From the Palette, drag a DataGrid widget from the Palette and drop it in your Layer widget (SearchListLayer) below the Panel widget (searchPanel).
2. Save the project.

The DataGrid doesn't have any columns yet. We'll set those up after we tie the DataGrid widget to the list of customers. For that, we need to import the database ("Import the Database" on page 14).

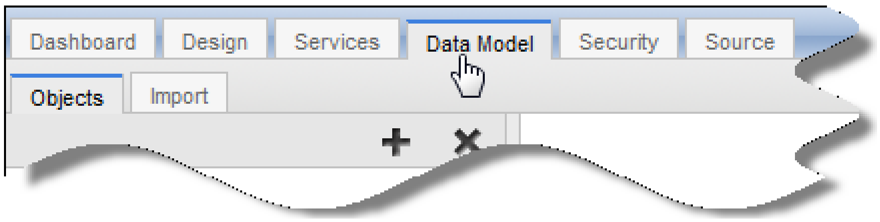
Import the Database

In this example, we'll use an HSQLDB sample database that is included in your WaveMaker Studio installation. The Sample database is called "cmdb" and it is located under the WaveMaker installation directory under **Support\Data**. For example, if WaveMaker is installed in the **C:\Program Files** directory, you would find the sample database directory in:

C:\Program Files\WaveMaker\Support\Data\cmdb

To import the database, follow these steps:

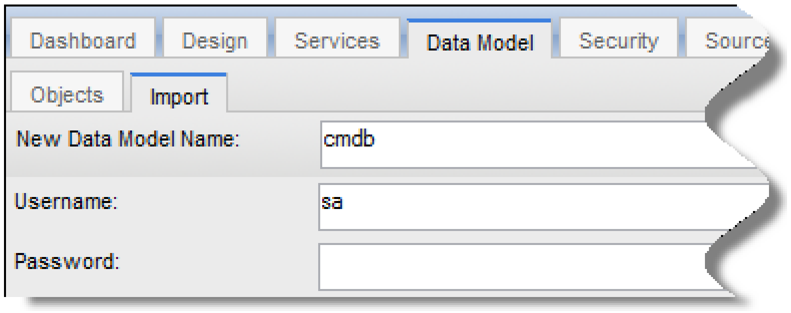
1. First open the Data Model Editor. In the Editor Tabs (across the top left of the WaveMaker Studio) click the **Data Model** tab.



2. The Data Model editor appears. The Data Model editor has an **Objects** tab and an **Import** tab.

- The **Objects** tab shows all the Data Objects currently available to your project. When the Objects tab is selected, the right side of the The Data Model editor shows information about the selected Data Object. Currently we don't have any Data Objects to select, so this part of the screen is blank.
- The **Import** tab shows the Import Database screen.

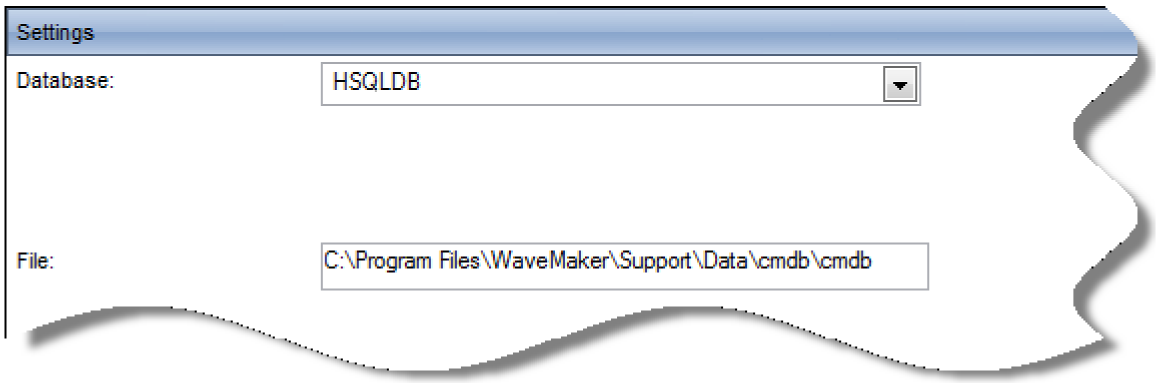
3. Click the **Import** tab. The Import Database screen appears. This screen contains fields for configuring the connection to the database you want to import. At the top of the screen, fill in the following fields:
   - **New Data Model Name**: Use this field to name the Data Model. The name can contain only alphanumeric characters. For this example, type in:
     `cmdb`
   - **Username**: This is where you type in a valid username for the database you're importing. In this example, type in:
     `sa`
   - **Password**: This is where you type in the password for the database. In this example, there is no password so delete the characters in this field, if any.



New Data Model Name: cmdb
Username: sa
Password:

4. In the "Settings" area of the Import Database screen, you select your database system and provide connection information. The connection information is different for different database systems. Fill in the "Settings" fields as follows:
   - **Database System:** The type of database you want to import. Choose **HSQLDB**. When you select the type of database, the "Settings" fields change to reflect the configuration requirements for whatever database system you selected.
   - **File**: The location of the cmdb database on your file system. For example, if your WaveMaker home directory is in C:\Program Files\WaveMaker, then you would type in:

`C:\Program Files\WaveMaker\Support\Data\cmdb\cmdb`

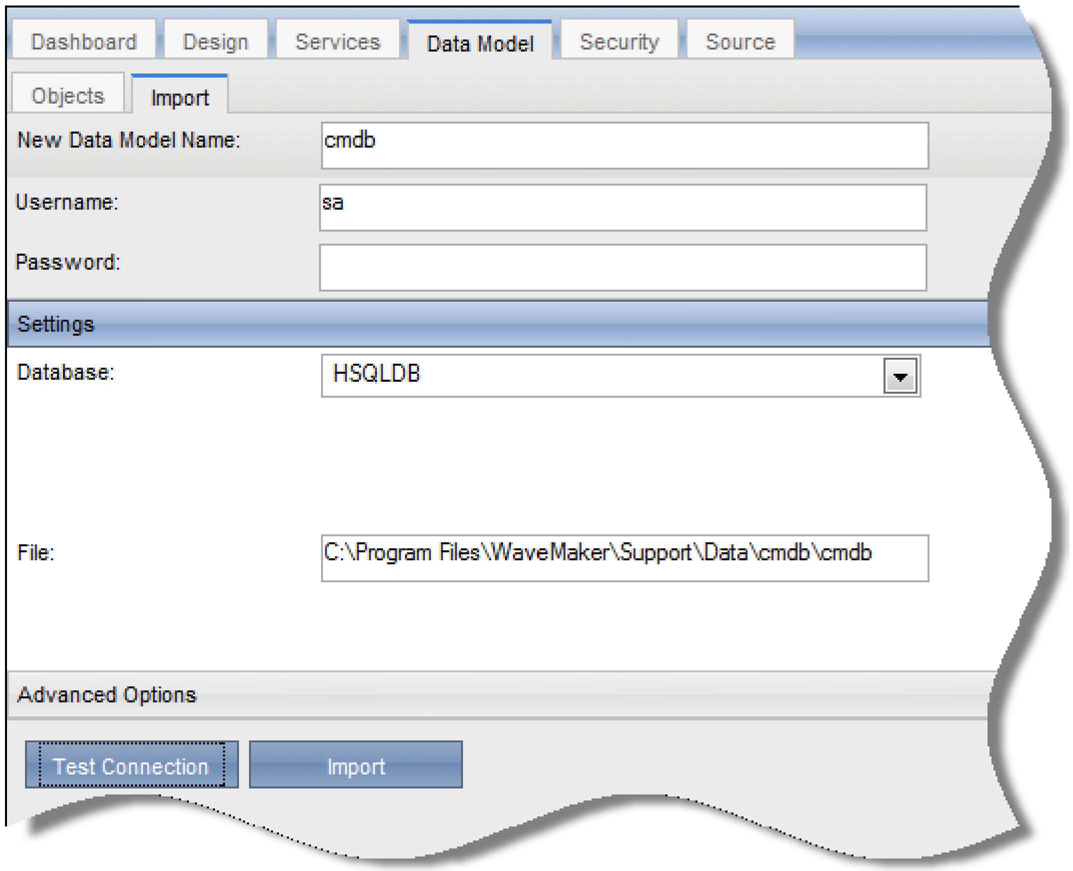


Settings

Database:	HSQLDB
File:	C:\Program Files\WaveMaker\Support\Data\cmdb\cmdb

5. Click the **Test Connection** button. You should get a "Connection Successful" message. If the test fails, go back and make sure your settings are correct.

   The completed Import Database form should look something like this:



New Data Model Name:	cmdb
Username:	sa
Password:	

Settings

Database:	HSQLDB
File:	C:\Program Files\WaveMaker\Support\Data\cmdb\cmdb

NOTE: The Import Database screen also has an "Advanced Options" menu, but we will do not need to use these advanced settings in this tutorial.

6. Click the **Import** button. WaveMaker Studio begins importing the database.

When you import a database, WaveMaker Studio automatically creates the data objects and service operations that allow your application to interact with that database. The service operations are listed on the **Services** tab and the data objects are listed on the **Data Model** tab.

7. Save the project.

Now that we have imported the database, we can set up the Service Calls and wire up the button and list ("Add Services to Call the Database" on page 17).

Add Services to Call the Database

For the search and list functionality, the application will need to call the Server and get data from the database. To do this, we set up a Service Call that gets the list of customers from the server. You set up the Service Call from the **Design** tab.

We want a Service Call that will perform a search on whatever text the user types into the input field and send back a list of customers with last names that match the text. To do this, we will call the **getCustomerList** operation from the cmdb service. WaveMaker Studio created the cmdb service automatically when we imported the cmdb database. The cmdb service automatically contains the basic operations you use to interact with the database.

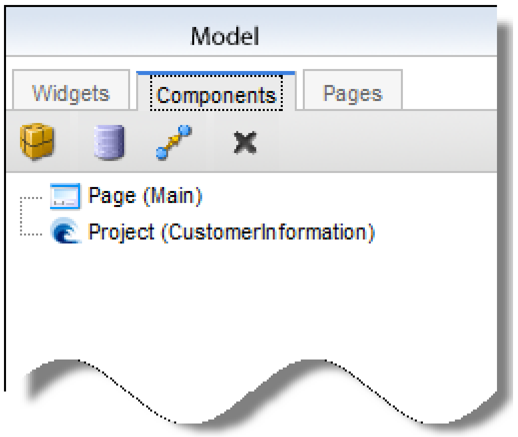
Click the **Services** tab to see the cmdb service. The **Services** tab allows you to edit and add services. Expand the cmdb service to see the operations it provides. In the list of operations, find the **getCustomerList** operation and click to expand that.

There are two inputs for this operation. The one we're interested in is the first one, searchInstance. What we want to do in our application is to take whatever value the user types into the input field and use that value for the customerName string in the searchInstance.

To set up this Service Call, follow these steps:

1. Click the **Design** tab to open the Page Designer.
2. In the Model Tree on the right side of the Page Designer, click on the **Components** tab. The **Components** tab shows you the Variables and services for your application and the different places where you can put them. Because this application has only one page, "Main", you have only one page on which to place components.

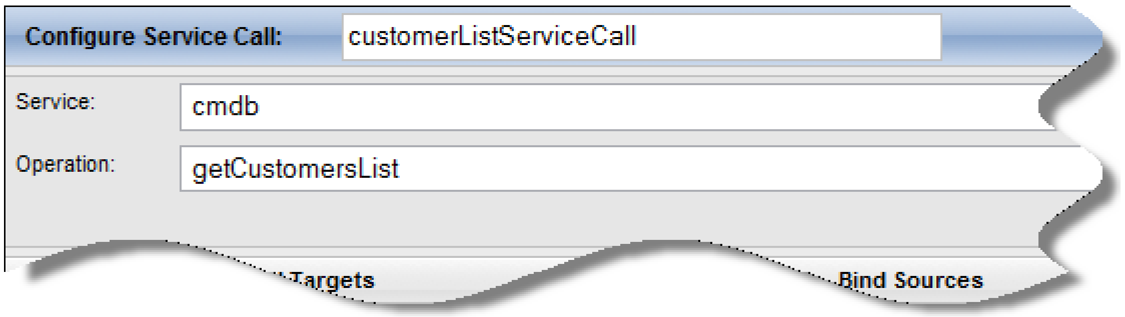
   You can alternatively put components on the Application level, however we do not need to do this in this example. By default, components are added to the current page.



3. Click on the cylinder icon to create a new Service Call scoped at the page level. The "Configure Service Call" dialog appears.
4. At the very top of the dialog, the Configure Service Call text field shows the name of the Service Call. All Service Calls are automatically named for you. It should say something like "serviceCall1". Change the name to:

   ```
   customerListServiceCall
   ```

   Note: Make Service Call names as descriptive as possible because you often need to select a particular Service Call from a list of other calls. I
5. From the **Service** drop-down menu, select **cmdb**.
6. Once you select a service, the operations for that service populate the **Operation** drop down. Select the **getCustomersList** operation. The top half of the Configure Service Call" dialog should now look like the following:

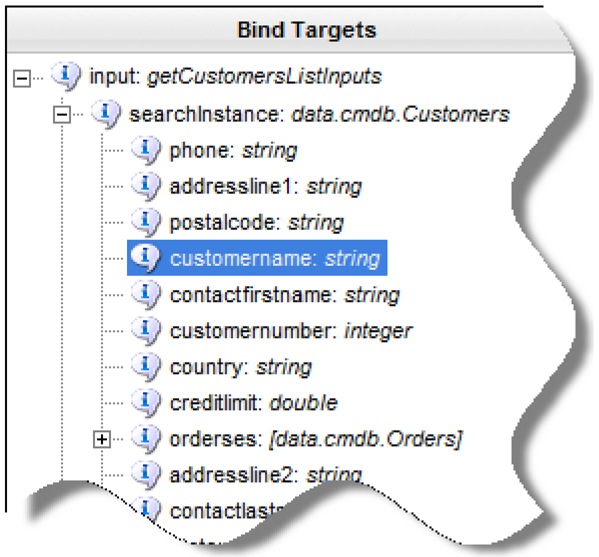


Configure Service Call: customerListServiceCall

Service: cmdb

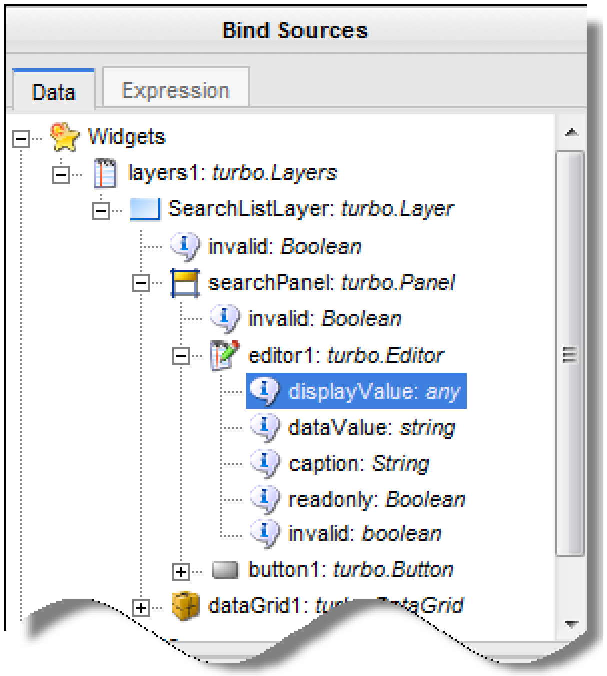
Operation: getCustomersList

7. Next we need to set up a binding so that the value in the input field (the Editor widget) gets sent as the input to the **getCustomersList** operation. In the "Bind Targets" list, click to expand the **searchInstance** input. We want to search against the customer's last name, so click on the **customername** input.

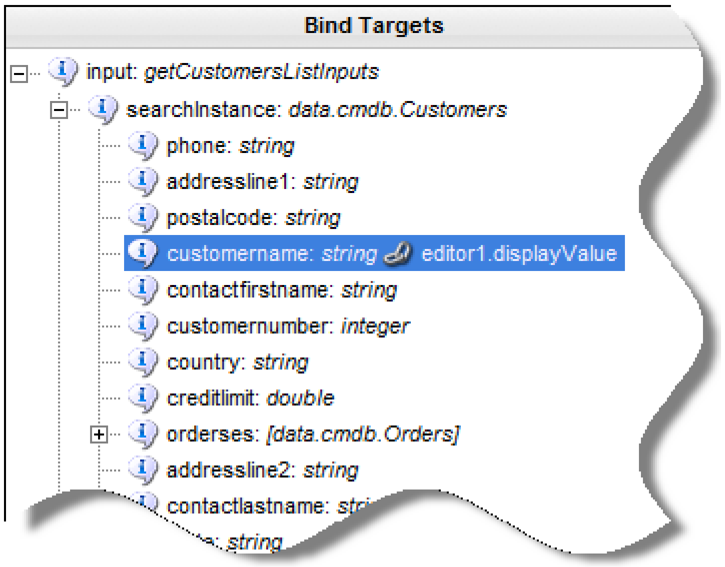


8. Next, in the “Bind Sources” list, we select the source for the customer’s last name. In this case we want that Editor input that we created in the search panel. Click to expand the container widgets until you see the Editor widget.

9. Expand the Editor widget and select the **displayValue** string.



10. A link icon appears over in the Bind Targets section of the screen. This shows that the binding is complete.


Bind Targets

input: *getCustomersListInputs*
searchInstance: *data.cmdb.Customers*
phone: *string*
addressline1: *string*
postalcode: *string*
customername: *string* editor1.displayValue
contactfirstname: *string*
customernumber: *integer*
country: *string*
creditlimit: *double*
orderses: *[data.cmdb.Orders]*
addressline2: *string*
contactlastname: *stri*
string

11. Click **Close**.

12. Save the project.

Now that the Service Call is complete, we can set up the Button and the DataGrid widgets ("Configure the Controls" on page 20).

Configure the Controls

Now that we have Service Calls defined in the application, we can wire up the controls so that the Button and DataGrid widgets work:

- "Configure the DataGrid Widget" on page 20
- "Configure the Search Button" on page 23

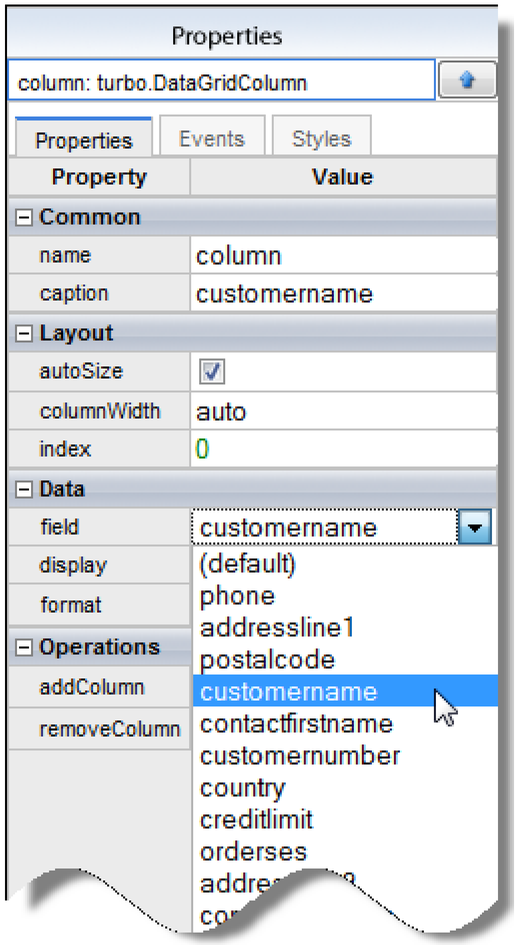
Configure the DataGrid Widget

To configure the DataGrid widget, follow these steps:

1. Select the DataGrid widget in the Page Designer.
2. With the DataGrid widget selected, on the **Properties** tab, click the button for the **binding** property. The Data Binding dialog appears. This screen allows you to bind items in your page to services and data.
3. Under "Bind Targets", select **dataSet** under the DataGrid widget. With the data set selected, we can now set the binding source.
4. Under "Bind Sources", select your **customerListServiceCall**. This is the Service Call that we just created.
5. Click **Close**.

6. Next we specify which fields the DataGrid should display. We can either add columns for each field we want or we can use the **autoColumns** operation to automatically generate DataGrid columns for each column in the table. In this example, we are going to configure three columns manually. Start by selecting the DataGrid in the Page Designer.

7. The DataGrid automatically has contains a single column, called "Cell 0". Select that column. (If you can't select it in the Page Designer, select it in the Model Tree.) In the Property Editor, under the "Data" heading select the **customername** option from the **field** property drop-down menu.



Property	Value
Common	
name	column
caption	customername
Layout	
autoSize	✓
columnWidth	auto
index	0
Data	
field	customername
display	
format	
Operations	
addColumn	
removeColumn	

The menu for the **field** property contains all the fields in the dataSet for this DataGrid. In this case, we set up the dataSet to be the Customer table in our cmdb database.

8. When you select **customername** for the **field** value, the DataGrid column name changes to match. Let's change the name of the DataGrid column. In the Property Editor, under the "Common" heading, change the value of the **caption** property to:

```
Customer Name
```

9. Next we add a second column to the DataGrid widget. In the Property Editor, under the “Operations” heading, click the button for the **addColumn** operation. A second column appears in the DataGrid widget.

10. Select the new column. (If you can’t select it in the Page Designer, select it in the Model Tree.) In the Property Editor, under the “Data” heading select **city** from the **field** property drop-down menu.

11. In the Property Editor, under the “Common” heading, change the value of the **caption** property to:

    ```
    City
    ```

12. Finally, we add a third column to the DataGrid widget. In the Property Editor, under the “Operations” heading, click the button for the **addColumn** operation. A third column appears in the DataGrid widget.

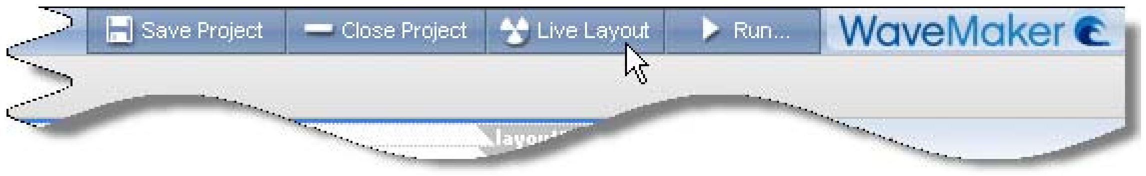
13. Select the new column. (If you can’t select it in the Page Designer, select it in the Model Tree.) In the Property Editor, under the “Data” heading select **state** from the **field** property drop-down menu.

14. In the Property Editor, under the “Common” heading, change the value of the **caption** property to:

    ```
    State
    ```

The basic DataGrid functionality is now configured. To see it work in the Page Designer, try the WaveMaker Studio LiveLayout™ feature. LiveLayout allows you to dynamically test run the application as you assemble it. To see it work, follow these steps:

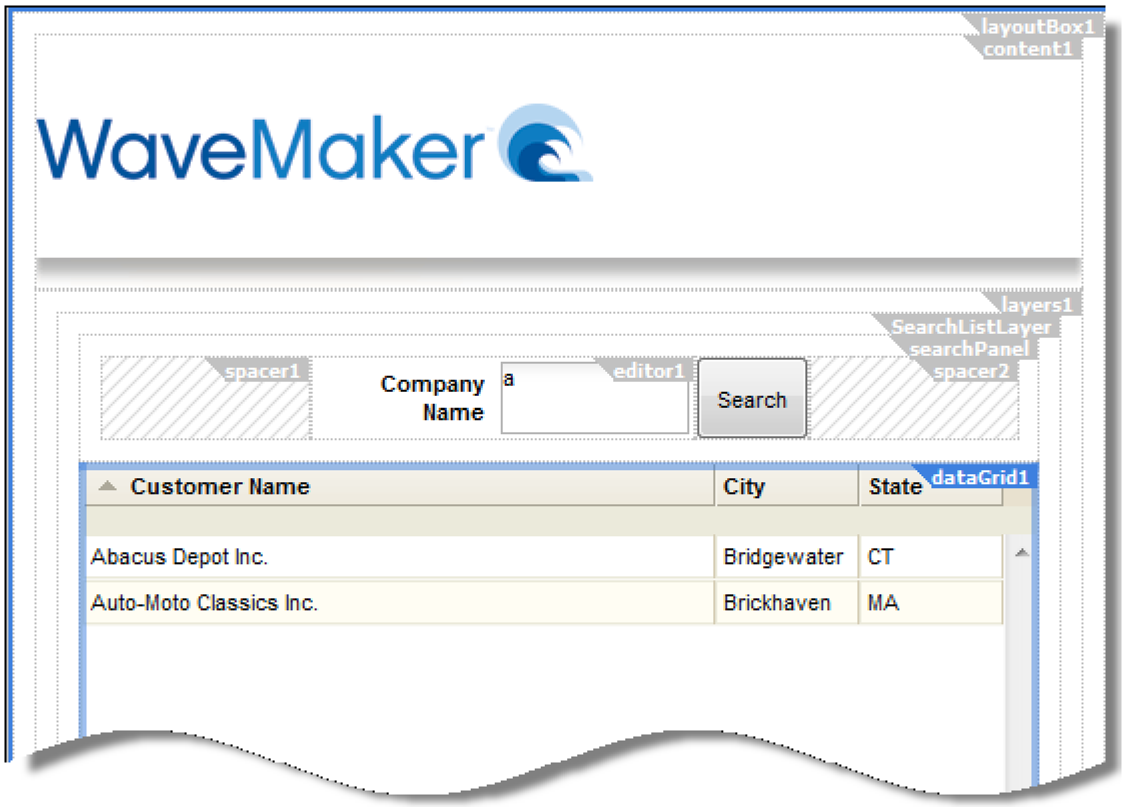
1. At the top right of WaveMaker Studio, in the Project tabs, click the **Live Layout** button.



2. Select the Editor widget in the Page Designer.
3. On the **Properties** tab, under “Editing” type in the following value for the **displayValue** property:

   a

4. In the Page Designer click to select the DataGrid widget.
5. On the **Properties** tab, under “Other”, click the **updateNow** button. The DataGrid now shows the data for this search.



Company Name: a

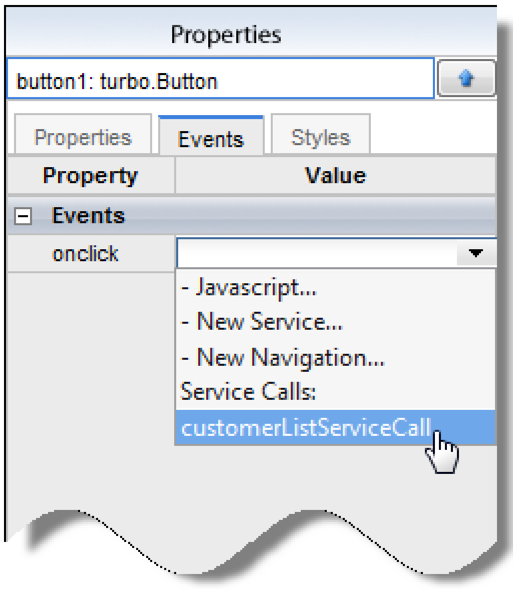
Customer Name	City	State
Abacus Depot Inc.	Bridgewater	CT
Auto-Moto Classics Inc.	Brickhaven	MA

The next step is to "Configure the Search Button" on page 23.

Configure the Search Button

We want to set up the Button widget to perform a search on the Customers table in the cmdb database. The search should take the text that the user typed into the text input box and search for that text against the customername column in the Customers table:

1. Select the Button widget in the Page Designer.
2. Click the **Events** tab (located next to the **Properties** tab). Click the arrow next to **onclick** property. A drop-down menu appears, listing the different ways you can wire up this button. Choose your **customerListServiceCall** Service Call.

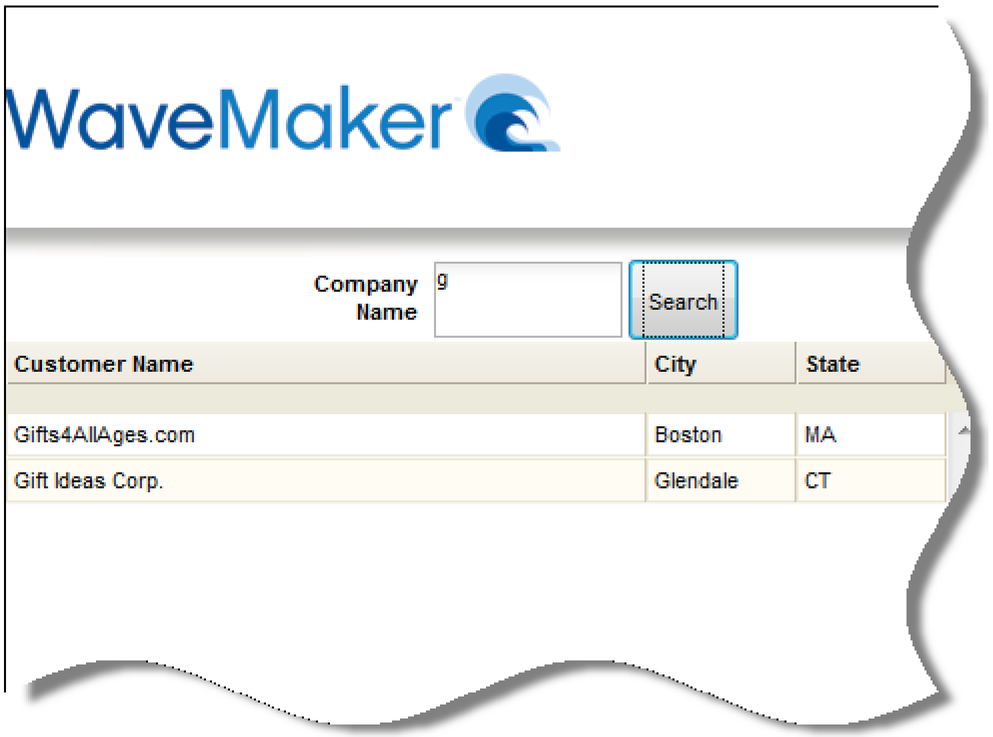


3. Save the project.

Run the project ("Run the Project" on page 24).

Run the Project

To try out the project, click the **Run** button on the upper right of the WaveMaker Studio screen. It might take a little while for WaveMaker Studio to build the preview. When it's finished, you can try your application.



g

Customer Name	City	State
Gifts4AllAges.com	Boston	MA
Gift Ideas Corp.	Glendale	CT

NOTE: Your browser might block this test run as a popup. Be sure to allow pop-ups from localhost to enable test runs.

Congratulations!

You have built your first WaveMaker Studio application. The next tutorial ("Add a Customer Detail Tab" on page 25) shows you how you to add a detail panel to the application!

Add a Customer Detail Tab

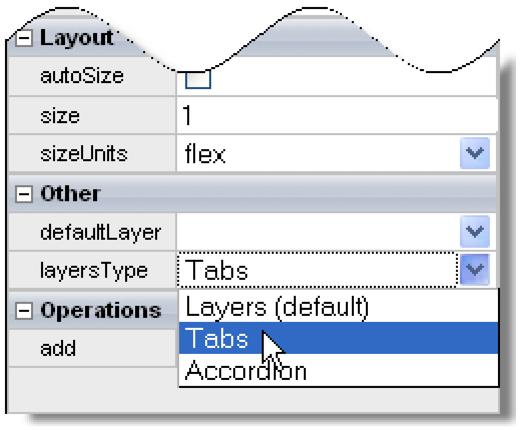
We want to add a tab that displays the details about a specific customer. When a user clicks a customer in the DataGrid, the details about that customer should appear in the Customer Detail tab, which should then become the active tab. There are four steps to making this work:

- "Add a New Tab" on page 26
- "Add the Service Call and Navigation Component" on page 27
- "Add the Detail Panel" on page 32
- "Set up the DataGrid" on page 36

Add a New Tab

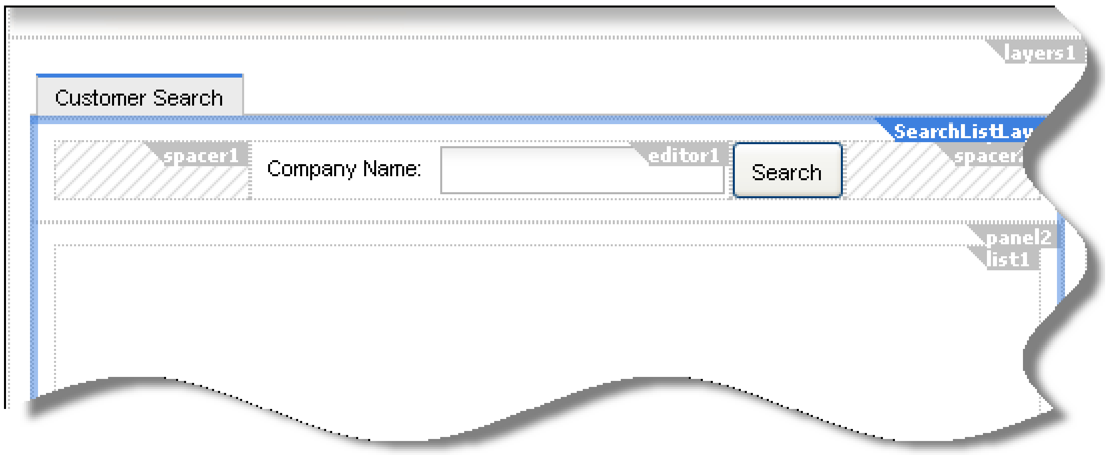
Instead of a single search/list page, we want to provide a page with two tabs: one for the search/list and one for details about the selected customer. The first step is to change to tabbed layers and to add a new tab:

1. If you aren't in the Page Designer, open the Main Page in the Page Designer now.
2. In the Model Tree, select the Layers widget.
3. Leave the Layers widget selected and go to the Property Editor. In the **Properties** tab, find the **layersType** property in the "Other" section of the Property list. From the **layersType** drop-down menu, select **Tabs**.



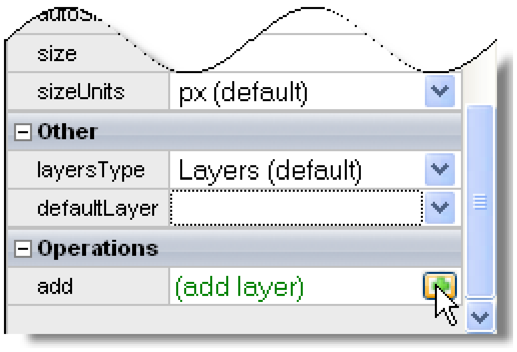
Layout	
autoSize	
size	1
sizeUnits	flex
Other	
defaultLayer	
layersType	Tabs
Operations	
add	

4. Your SearchListLayer layer changes to a tabbed layer. It still contains your search/list content, but now it has a tab. The label on the tab reads "layer1". We define the tab label by setting the **caption** property.
5. Select the SearchListLayer.
6. In the list of "Other" properties, for the **caption** property type in: `Customer Search`
7. The tab should look something like this:



Now we're ready to add the new tab.

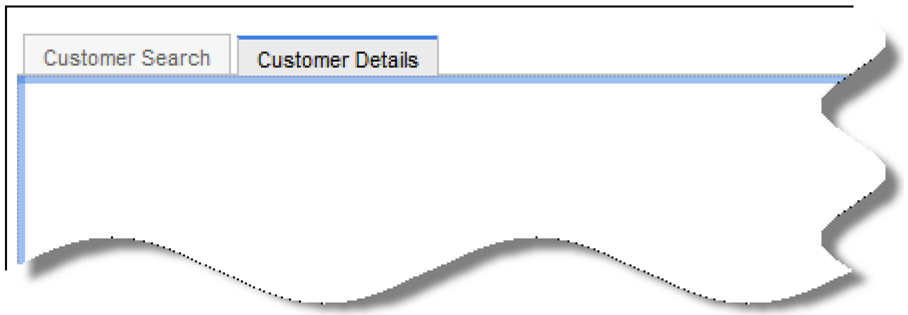
8. In the Model Tree, select the Layers widget again.
9. With the Layers widget selected, go to the Property Editor and scroll down to the Properties section labeled "Operations". Click the "add layer" button on the **add** property. This adds a new layer.



size	
sizeUnits	px (default)
Other	
layersType	Layers (default)
defaultLayer	
Operations	
add	(add layer)

10. In the Model Tree, select the new layer. By default, the default name is "layer1". This is also the default caption. We want to change both.
11. With the new layer selected, go to the Property Editor.
12. In the list of "Common" properties, for the **name** property type in:
    `DetailLayer`
13. In the list of "Other" properties, for the **caption** property type in:
    `Customer Details`

    The tabs should now look like this:



14. Save the project.

Now that the tabs are set up, we still need to set up the Service Call and the content on the detail layer. Let's start with the Service Call ("Add the Service Call and Navigation Component" on page 27).

Add the Service Call and Navigation Component

For the detail functionality, we need two things:

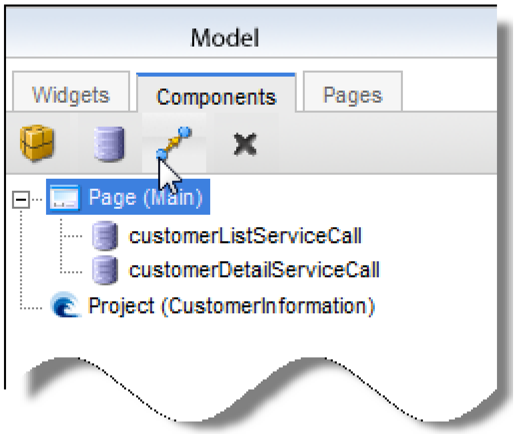
- We need a call that gets the details about whatever customer is selected in the DataGrid ("Add the Call to Get the Customer Details" on page 29).
- We need a *Navigation component* that reveals the Details layer ("Add the Call to Get the Customer Details" on page 29).

Add the Navigation to Show the Details Layer

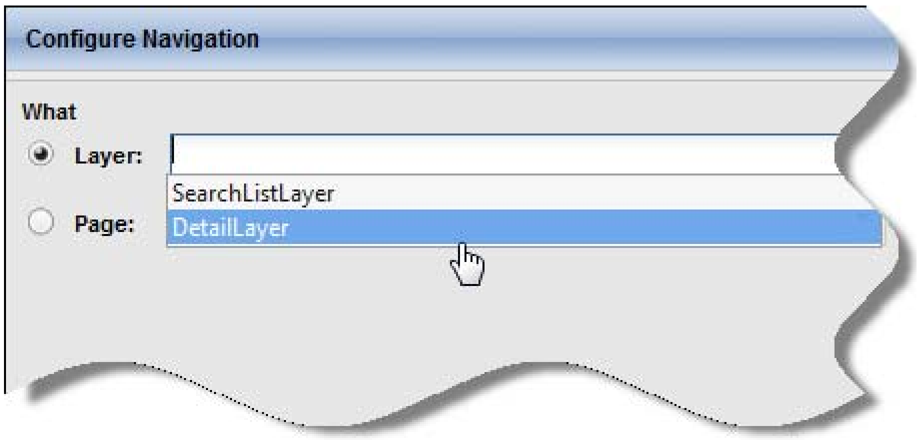
When a user clicks a record in the customer DataGrid, we want the details about that customer to appear in the Customer Detail tab. In order to make this work, we need to set up a *Navigation component*. A Navigation component is like a Service Call, except that it stores a navigation configuration, rather than a service operation configuration.

The Navigation component says what you want to show the user (which page or layer) and where you want to show it (this is an option only for pages.) To create the new Navigation component, follow these steps:

1. Open the Page Designer.
2. In the Model Tree, click on the **Components** tab.
3. Click on the New Navigation icon.



4. The Configure Navigation screen appears. This screen allows you to choose whether you want to navigate to a page or a layer. In this example, we want to navigate to a layer. From the **Layer** drop-down menu, select the **DetailLayer** option.



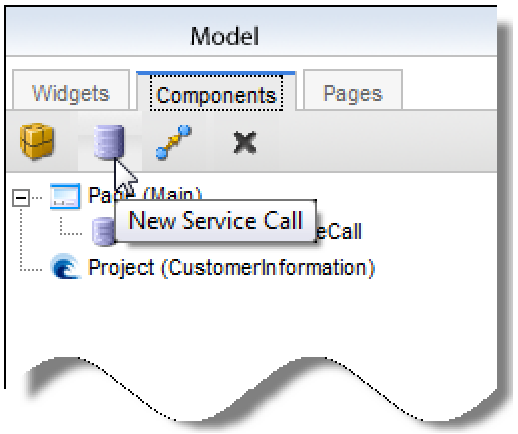
5. Click **Save**.

That's it. The next step is to "Add the Call to Get the Customer Details" on page 29.

Add the Call to Get the Customer Details

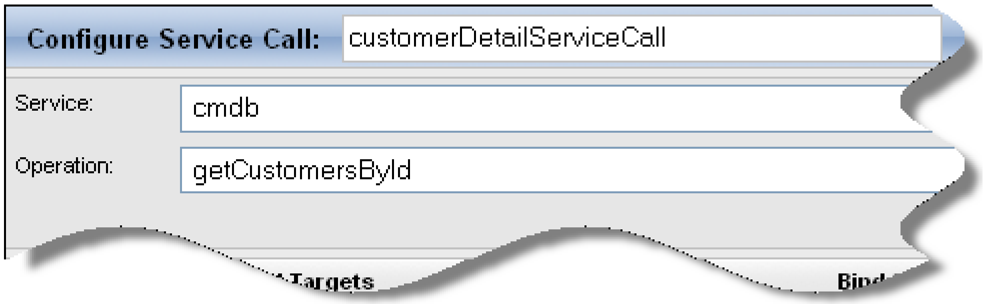
For the customer details we need a Service Call that takes the ID of the selected customer in the DataGrid and returns all the information about that customer. To do this, we will call the **getCustomersById** operation from the cmdb database service. To set up this Service Call, follow these steps:

1. Open the Page Designer.
2. In the Model Tree on the right side of the Page Designer, click on the **Components** tab.
3. Click on the cylinder icon to create a new Service Call scoped at the page level.



4. The "Configure Service Call" dialog appears.

5. At the very top of the screen there is a text field that contains the name of the Service Call. All Service Calls are automatically named for you, but we want something more descriptive. Change the name to: customerDetailServiceCall
6. From the **Service** drop-down menu, select **cmdb**.
7. Once you select a service, the operations for that service populate the **Operation** drop down. Select the **getCustomersById** operation.



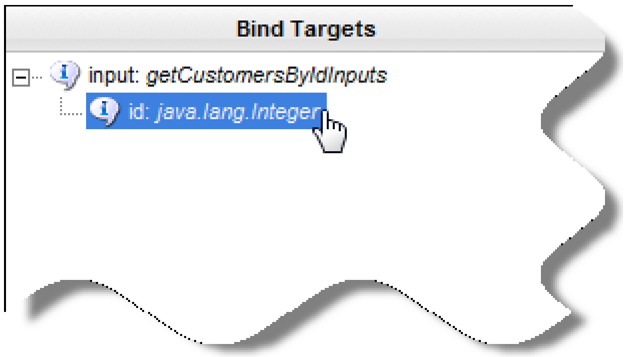
customerDetailServiceCall

Service: cmdb

Operation: getCustomersById

Targets

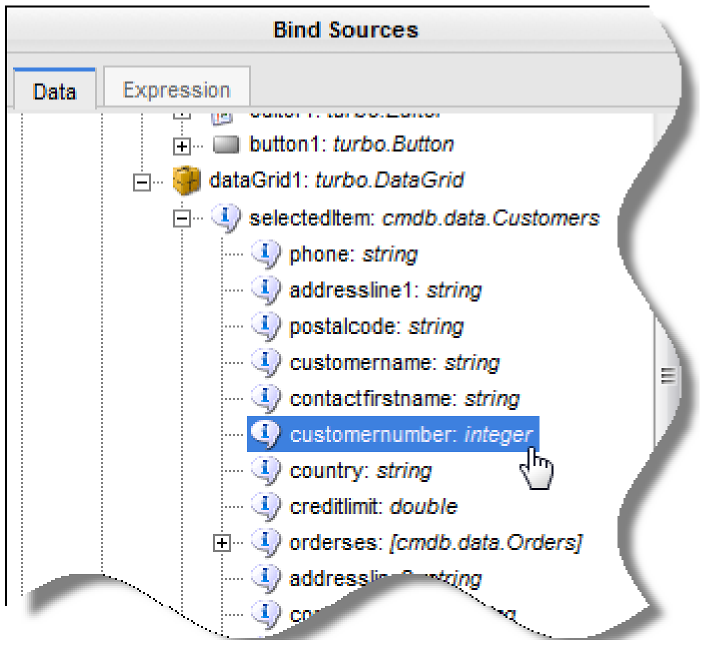
8. Next we need to set up a binding so that the input to the **getCustomersList** operation is the ID of the selected item in the DataGrid. In the "Bind Targets" list, click on the **id** input.



input: *getCustomersByIdInputs*

id: *java.lang.Integer*

9. Next, in the "Bind Sources" list, we select the source for the customer's ID. In this case we want the *selected* item in the DataGrid widget. Click to expand the container widgets until you see the DataGrid widget and then click to expand that to see the selectedItem object.
10. Expand the selectedItem and select **customernumber**.

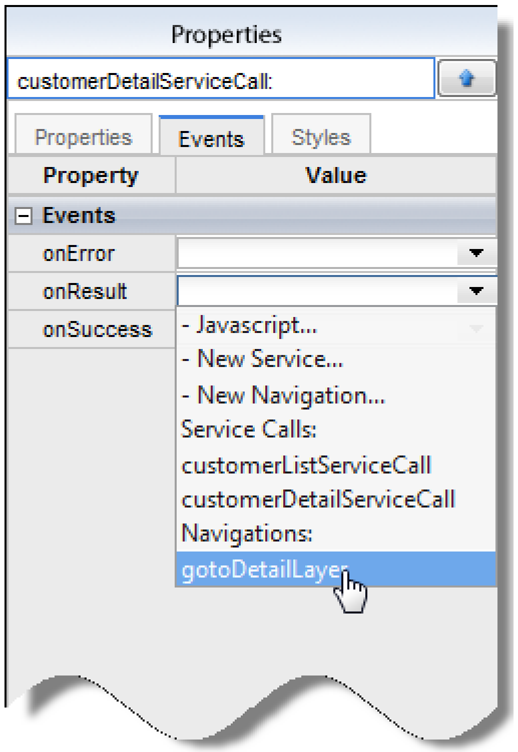


11. A link icon appears over in the Bind Targets section of the screen. This shows that the binding is complete.

12. Click **Close**.

13. As it stands now, this Service Call gets the customer details, but does not actually show them because the Details tab is not yet visible. We can fix that using the onResult event. Leave the **customerDetailServiceCall** selected in the Model Tree and in the Property Editor, click the **Events** tab.

14. Open the menu for the **onResult** event. The menu includes all the Service Calls and Navigation components you have created. Select the **goToDetailLayer** Navigation component that you created earlier (this is probably at the bottom of the list).



NOTE: The onResult menu allows us to create a new Service Call or Navigation component, so we could have just as easily created our Navigation component for here.

15.Save the project.

Now we can set up the detail panel ("Add the Detail Panel" on page 32).

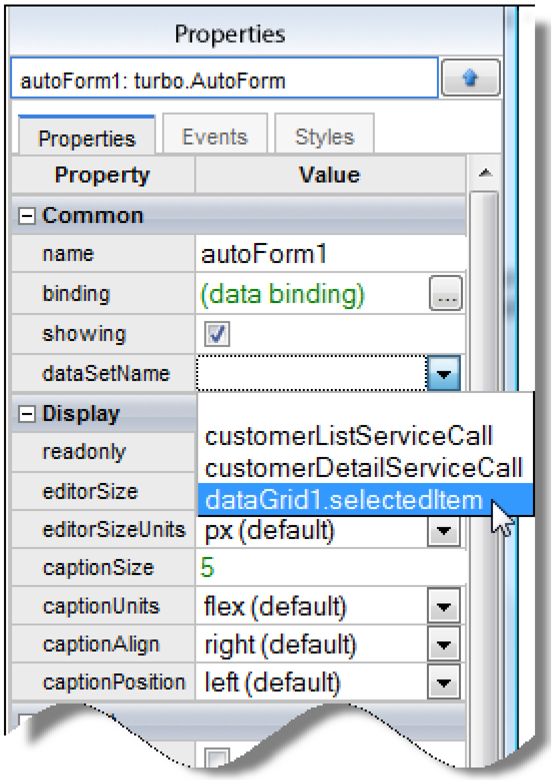
Add the Detail Panel

To show details about whatever customer is selected in the DataGrid widget, we use an AutoForm widget. Follow these steps:

1. If you aren't in the Page Designer, open the Main Page in the Page Designer now.
2. In the **Widgets** tab of the Model Tree, select your DetailLayer.
3. Drag an AutoForm widget from the Palette and drop it inside the DetailLayer. (The AutoForm widget is located in the "Common" section of the Palette.)
4. In the Property Editor, open the menu for the **sizeUnits** property and select **flex**. By default, the **size** property is set to **1**.
5. Now we need to bind the AutoForm widget to the input data (in this case, the record selected in the DataGrid.) We could use the **binding** property to do this, but the AutoForm provides an easier way.

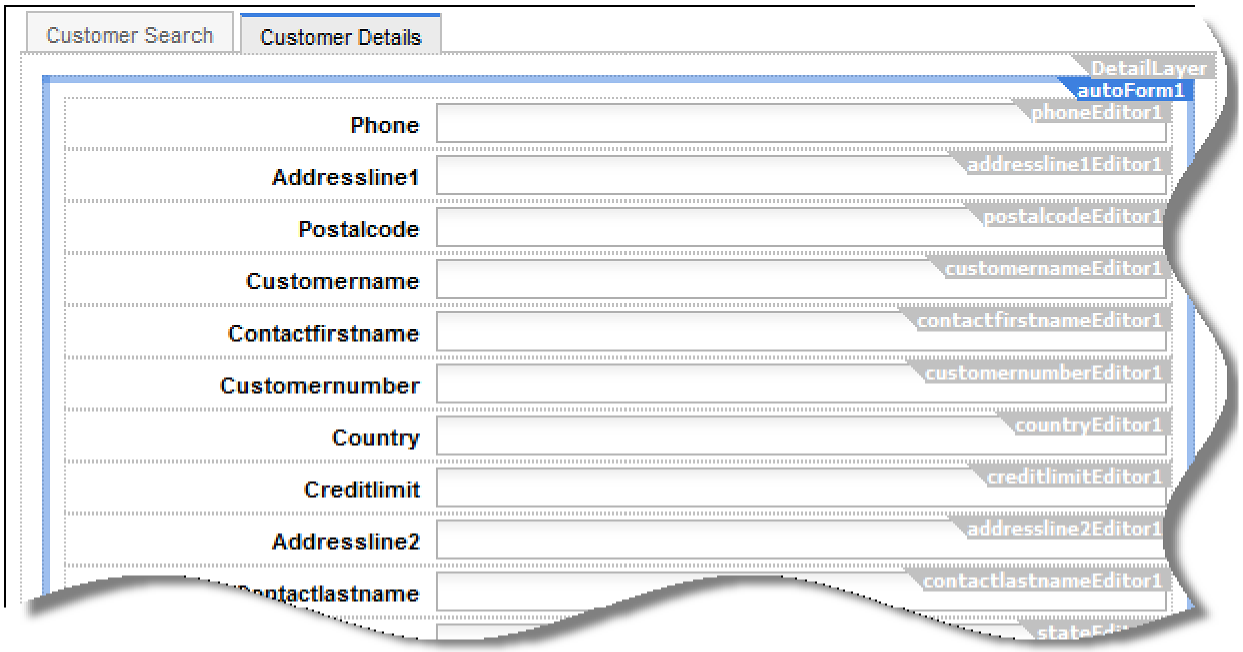
In the Property Editor, open the menu for the **dataSetName** property. The **dataSetName** property allows you to quickly choose a data set for the AutoForm widget. The **dataSetName** menu includes Service Calls and Variables and it also includes the “selected item” in DataGrid widgets.

6. From the **dataSetName** drop-down menu, select the **dataGrid1.selectedItem** option. This represents whatever item the user selects in the DataGrid widget.



Property	Value
Common	
name	autoForm1
binding	(data binding)
showing	✓
dataSetName	
Display	
readonly	
editorSize	
editorSizeUnits	px (default)
captionSize	5
captionUnits	flex (default)
captionAlign	right (default)
captionPosition	left (default)

When you select an item from the **dataSetName** property menu, the AutoForm widget automatically generates Editor widgets for every item in the DataSet (*except* lists and substructures). The binding on these Editor widgets is already done for you.

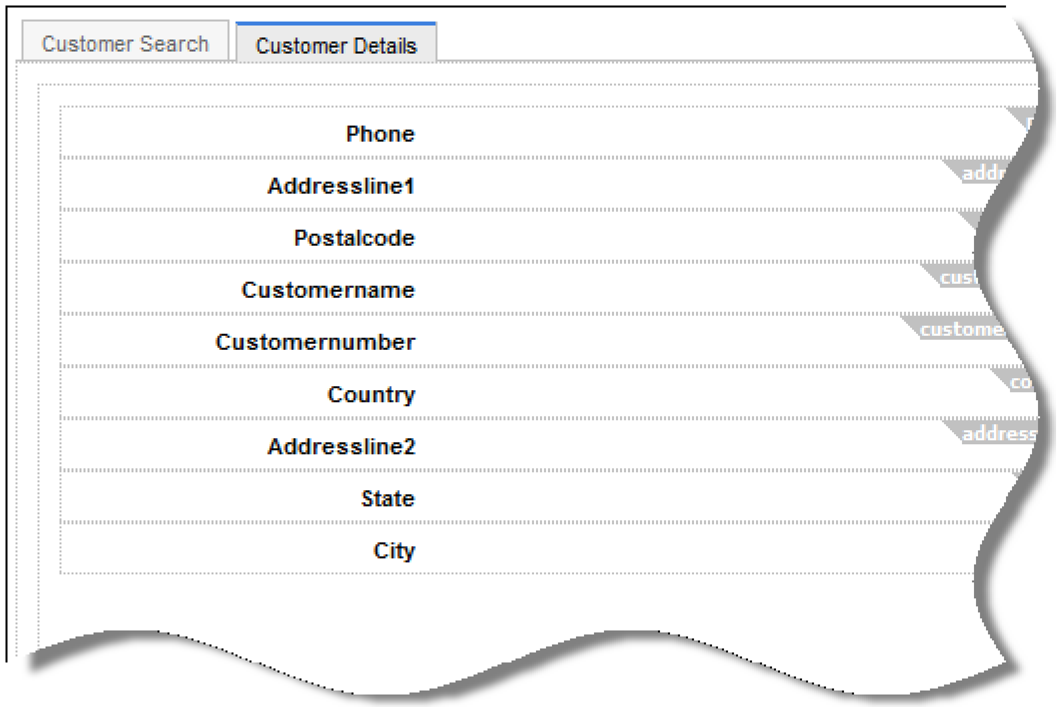


7. As it stands now, these Editors are set up as input fields. We want them to be output fields. To set this up, leave the AutoForm selected and go to the Property Editor.

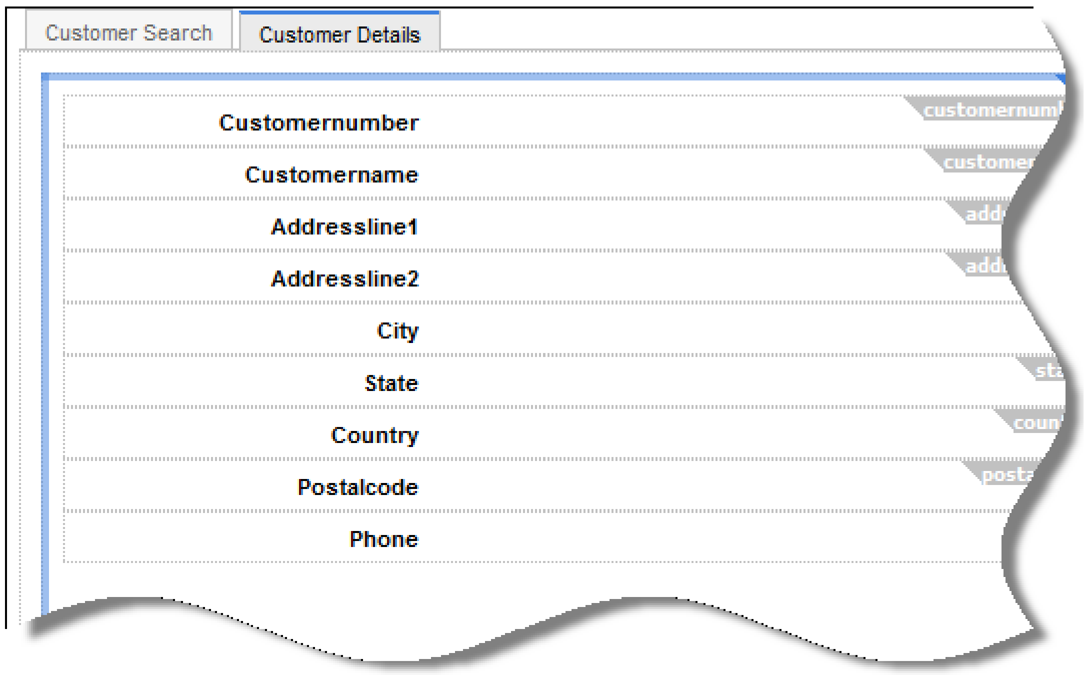
8. Click to check the **readonly** property (it's in the Property Editor under the "Display" heading.) Now all the fields are output fields.

9. By default, the AutoForm widget creates Editor widgets for every item in the DataSet, however we are free to delete any Editor widgets that we don't need. Select the Editor widget named "Salesrepemployeenumber" and click the Delete icon in the Design Toolbar.

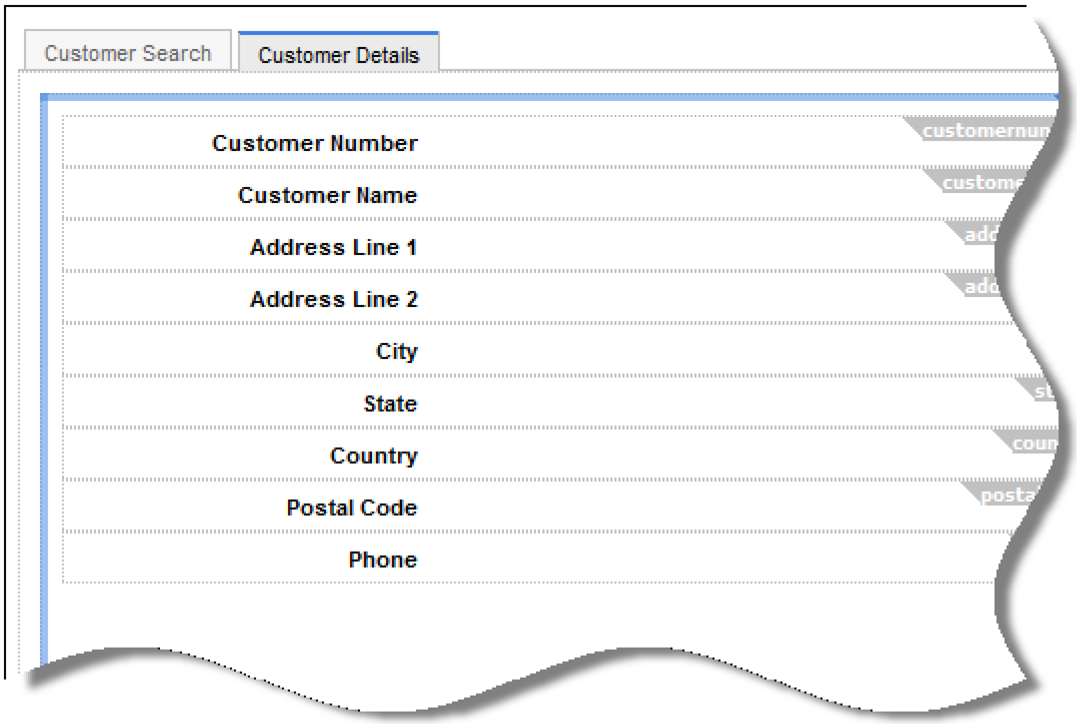
10. One-by-one select and delete the following Editor widgets: "Contactfirstname", "Contactlastname" and "Creditlimit". The AutoForm widget should now look something like this:



11. We can also move the Editor widgets around within the AutoForm. Drag the Editor widgets until they are in the following order, from top to bottom: “Customernumber”, “Customername”, “Addressline1”, “Addressline2”, “City”, “State”, “Country”, “Postalcode” and “Phone”.



12. We are also free to change each Editor widget's individual properties. In this example, we change the **caption** property for the following Editor widgets:
   - Change "Customernumber" to "Customer Number"
   - Change "Customername" to "Customer Name"
   - Change "Addressline1" to "Address Line 1"
   - Change "Addressline2" to "Address Line 2"
   - Change "Postalcode" to "Postal Code"



Customer Number
Customer Name
Address Line 1
Address Line 2
City
State
Country
Postal Code
Phone

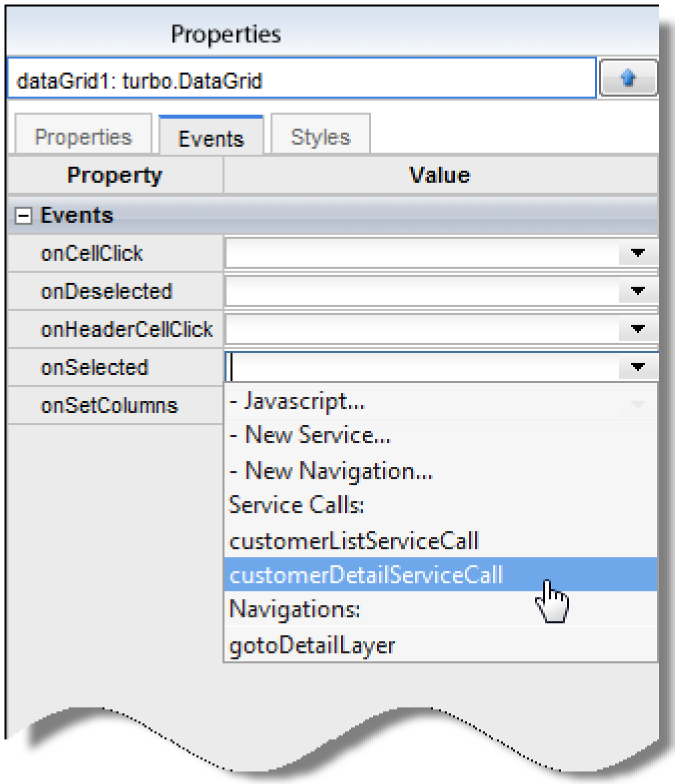
13. Save the project.

The last step is to "Set up the DataGrid" on page 36.

Set up the DataGrid

The final thing we want to do here is to set up the DataGrid widget so that when a user selects something, the AutoForm widget gets repopulated. To do that, follow these steps:

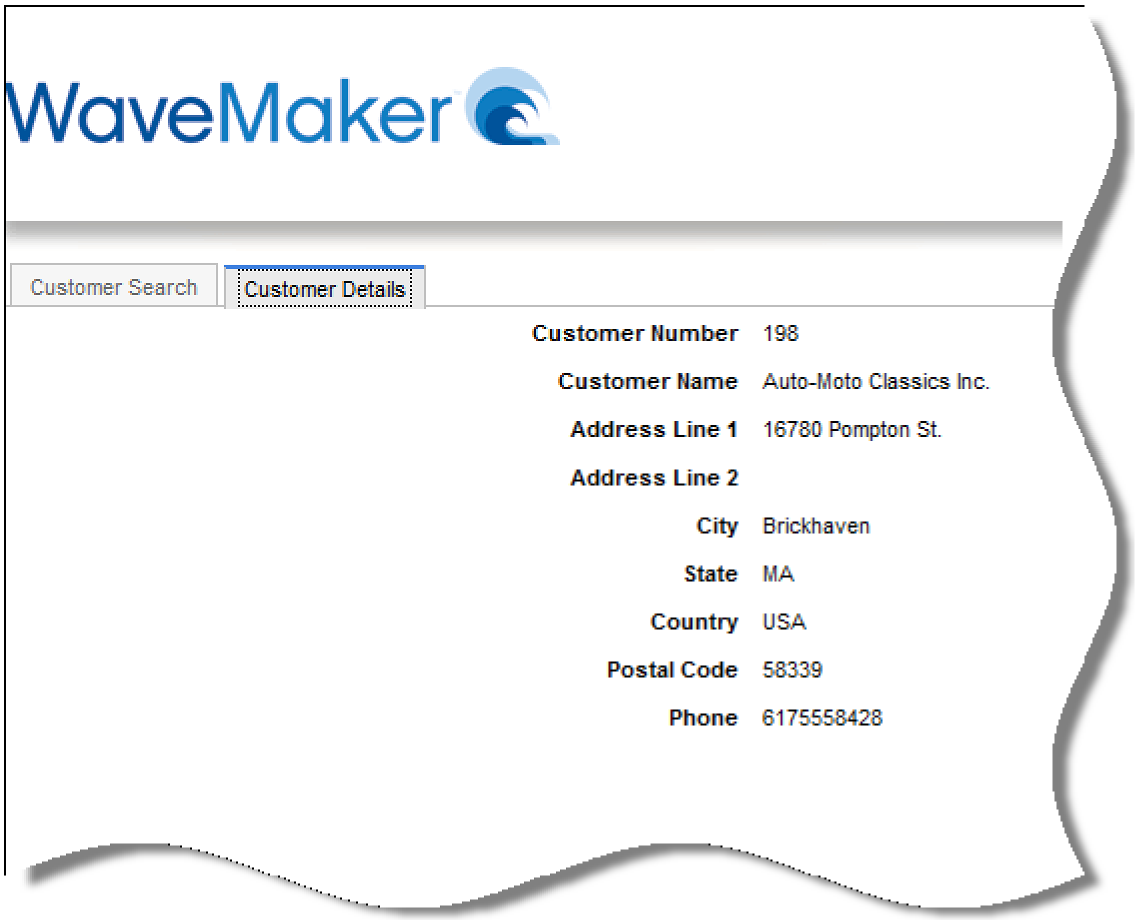
1. From the **Widgets** tab of the Model Tree, select the DataGrid.
2. Click on the **Events** tab.
3. Click on the button for the **onSelected** event.
4. From the drop-down menu, choose the **customerDetailServiceCall** option.



5. Save the project.

Run the Project

To try out the project, click the **Run** button on the upper right of the WaveMaker Studio screen. It might take a little while for WaveMaker Studio to build the preview. When it's finished, you can try your application. Try selecting an item in the DataGrid.



Customer Number	198
Customer Name	Auto-Moto Classics Inc.
Address Line 1	16780 Pompton St.
Address Line 2	
City	Brickhaven
State	MA
Country	USA
Postal Code	58339
Phone	6175558428

Click the Customer Search tab to go back to the search form. If this application didn't use tabbed layers, we would have needed a button or link on the detail layer so that the user could get back to the search/list layer.

Congratulations!

You have added tabbed layers to your application. The next tutorial ("Add a Web Service Call" on page 38**) shows you how you to add a Web Service Call!**

Add a Web Service Call

In this section we'll add a Web Service Call to verify the customer address shown on the Customer Detail tab. The basic steps are as follows:

1. "Import the Web Service" on page 39
2. "Add a Service Call" on page 41
3. "Add the Address Verification Button" on page 46

Import the Web Service

In this section, we're going to add a call to a Web Service to do a quick address verification. For this example, we'll use a StrikeIron USA Address verification SOAP service. SOAP services are defined by a WSDL file, which you will need in order to import the service.

The steps for importing the Web Service are:

1. "Locate the WSDL File" on page 39
2. "Import the SOAP Service" on page 39

Locate the WSDL File

The WSDL file for this service is called **USAddressVerification.wsdl** and it is included in your WaveMaker installation. It is located under the WaveMaker installation directory under **Support\Tutorial\soap**. For example, if WaveMaker is installed in the **C:\Program Files** directory, you would find the WSDL file in:

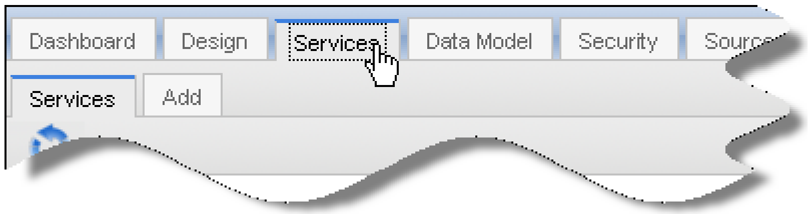
C:\Program Files\WaveMaker\Support\Tutorial\soap

Make a note of the path.

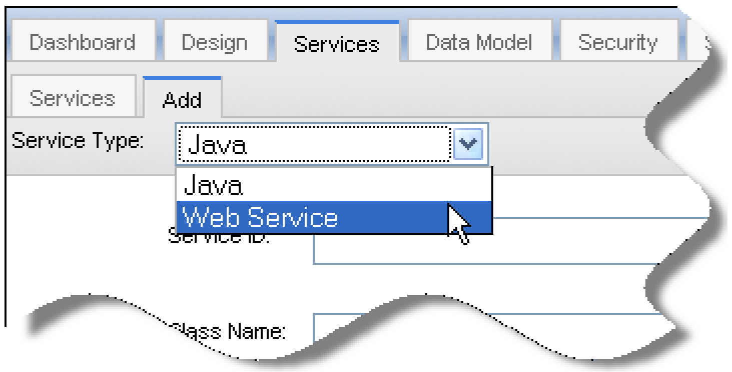
Import the SOAP Service

To import the Address Verification SOAP service, follow these steps:

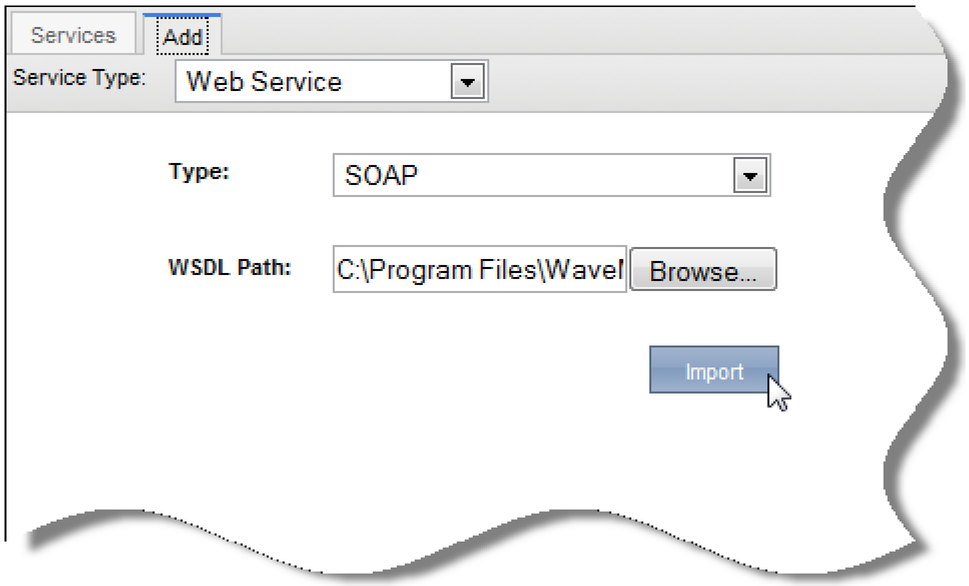
1. Click the **Services** tab in the Editor tabs across the top left of the WaveMaker Studio.



2. The Services Editor appears. Click the **Add** tab to add the new service.
3. From the **Service Type** drop-down menu, select the **Web Service** option.

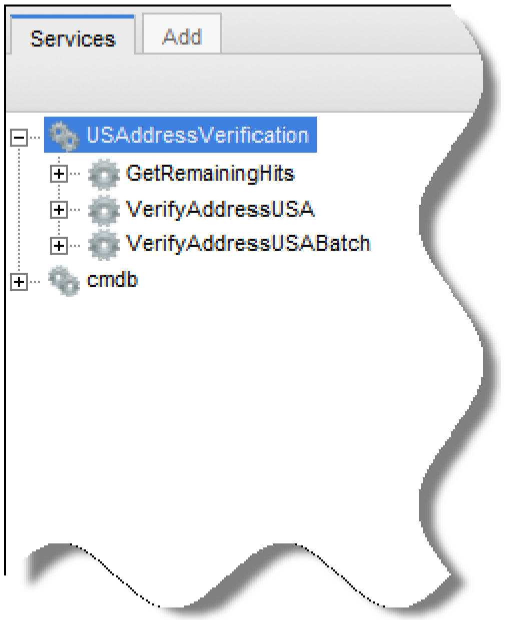


4. The Add Service screen now shows the fields you need to import a Web Service. Fill in the fields as follows:
   - ∞ **Service Type**: Select **SOAP** from the drop-down menu (this is the default).
   - ∞ **WSDL Path:** Click the **Browse** button. Browse to the location of the WSDL file ("Locate the WSDL File" on page 39).
5. Click the **Import** button.



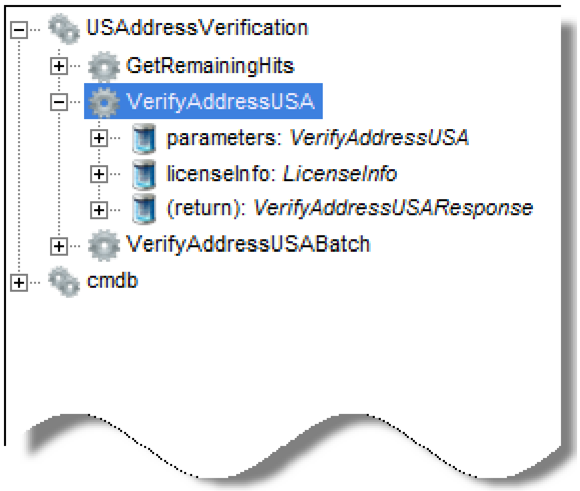
WaveMaker Studio imports the service.

6. Save the project.
7. Now the SOAP service is available to your application. Click to expand it in the **Services** tab.



Here you can see that the service has three operations: **GetRemainingHits**, **VerifyAddressUSA**, and **VerifyAddressUSABatch**.

8. Click to expand the **VerifyAddressUSA** operation. You can see the input parameters and returns for this operation. We'll need this information to set up our Service Call.


USAddressVerification
GetRemainingHits
VerifyAddressUSA
parameters: *VerifyAddressUSA*
licenseInfo: *LicenseInfo*
(return): *VerifyAddressUSAResponse*
VerifyAddressUSABatch
cmdb

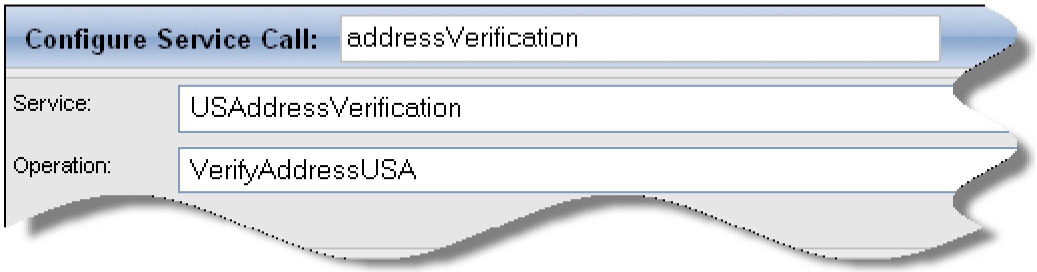
The next step is to "Add a Service Call" on page 41.

Add a Service Call

We want a Service Call that verifies the address of the customer currently listed in the Detail tab. To do this, we will call the **VerifyAddressUSA** operation from the USAddressVerification SOAP service that we just imported. To set up this Service Call, follow these steps:

1. Click the **Design** tab to open the Page Designer.
2. In the Model Tree on the right side of the Page Designer, click on the **Components** tab.
3. Click on the cylinder icon to create a new Service Call scoped at the page level. The "Configure Service Call" dialog appears.
4. At the very top of the screen there is a text field that contains the name of the Service Call. All Service Calls are automatically named for you, but we want to change the name to something more descriptive. Change the name to:

   `addressVerification`
5. From the **Service** drop-down menu, select **USAddressVerification**.
6. Once you select a service, the operations for that service populate the **Operation** drop down. Select the **VerifyAddressUSA** operation.

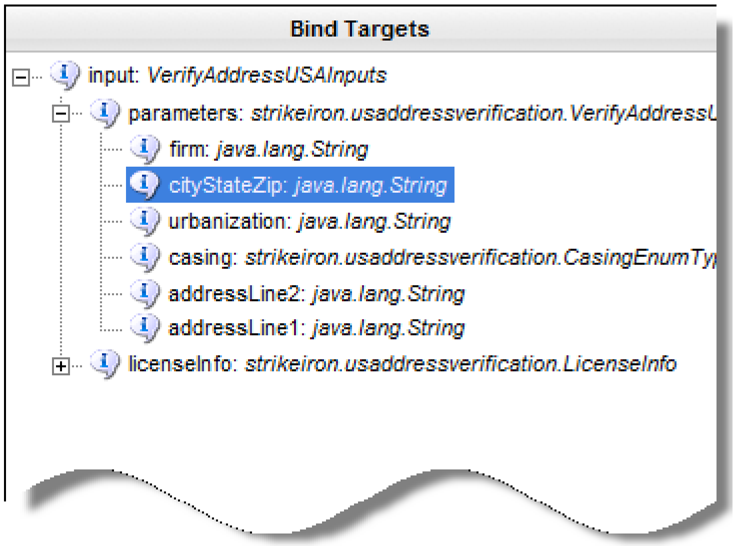


Configure Service Call: addressVerification

Service: USAddressVerification

Operation: VerifyAddressUSA

7. Next we need to set up a binding so that the value in the input field gets sent as the input to the **SOAP** operation. In the "Bind Targets" list, click to expand the **parameters** object.
8. Under parameters, click the **cityStateZip** input.



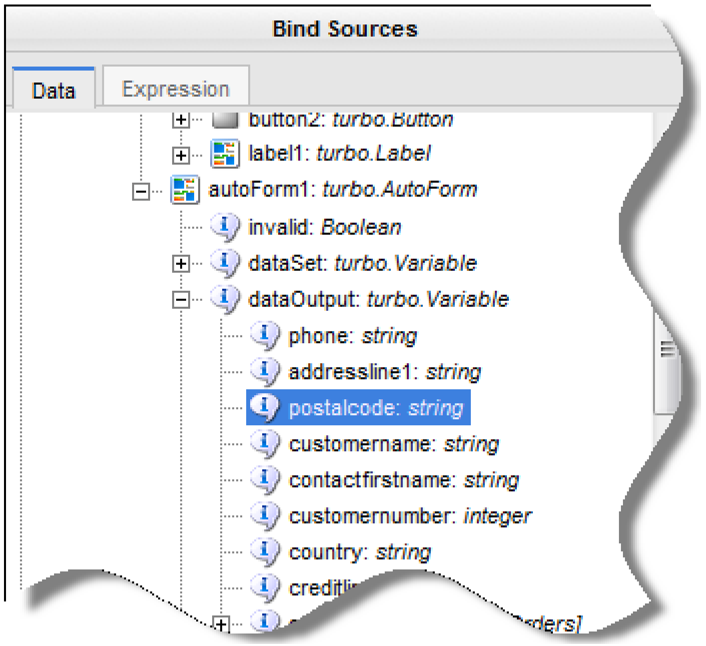
Bind Targets

```
input: VerifyAddressUSAInputs
  parameters: strikeiron.usaddressverification.VerifyAddressU
    firm: java.lang.String
    cityStateZip: java.lang.String
    urbanization: java.lang.String
    casing: strikeiron.usaddressverification.CasingEnumTy
    addressLine2: java.lang.String
    addressLine1: java.lang.String
  licenseInfo: strikeiron.usaddressverification.LicenseInfo
```

Next, in the "Bind Sources" list, we need to select a data source for the customer zip code. We want the zip code for the record displayed in the

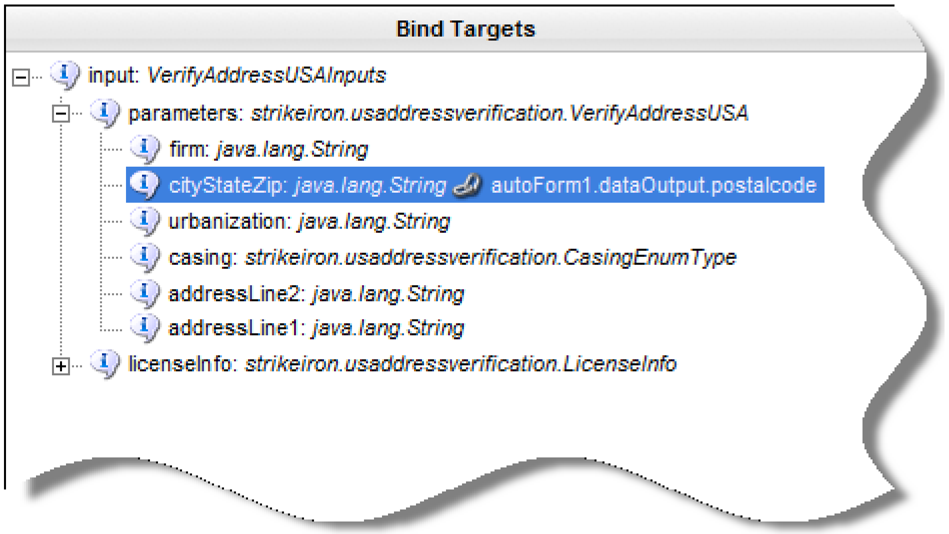
AutoForm widget. For this, we will use the AutoForm widget's "dataOutput" object. In the "Bind Sources" list, click to expand the container widgets until you see the AutoForm widget and then click to expand that to see the **dataOutput** object.

9. Expand **dataOutput** and select **postalcode**.



```
button2: turbo.Button
label1: turbo.Label
autoForm1: turbo.AutoForm
  invalid: Boolean
  dataSet: turbo.Variable
  dataOutput: turbo.Variable
    phone: string
    addressline1: string
    postalcode: string
    customername: string
    contactfirstname: string
    customernumber: integer
    country: string
    creditli
```

10. A link icon appears over in the Bind Targets section of the screen. This shows that the binding is complete.

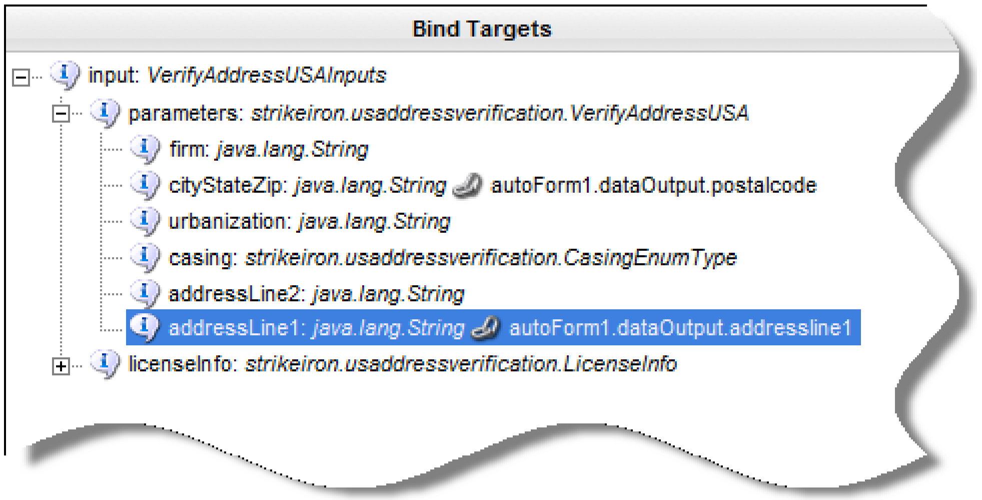


```
input: VerifyAddressUSAInputs
  parameters: strikeiron.usaddressverification.VerifyAddressUSA
    firm: java.lang.String
    cityStateZip: java.lang.String  autoForm1.dataOutput.postalcode
    urbanization: java.lang.String
    casing: strikeiron.usaddressverification.CasingEnumType
    addressLine2: java.lang.String
    addressLine1: java.lang.String
  licenseInfo: strikeiron.usaddressverification.LicenseInfo
```

11. In the same way, we'll set up a value for the **addressLine1** input parameter. In the "Bind Targets" list, under parameters, click the **addressLine1** input.

12. Next, in the "Bind Sources" list, we select the source for the address. In this case we want the **addressline1** field from the AutoForm **dataOutput** object. Expand **dataOutput** and select **addressline1**.

13. A link icon appears over in the Bind Targets section of the screen. Now two bind targets have bindings: **addressLine1** and **cityStateZip**.



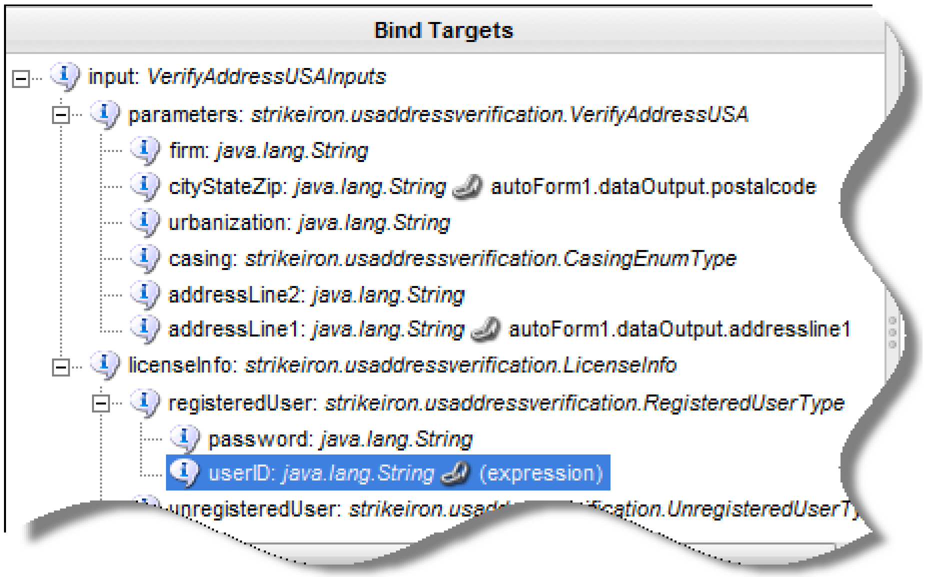
Bind Targets
```
input: VerifyAddressUSAInputs
  parameters: strikeiron.usaddressverification.VerifyAddressUSA
    firm: java.lang.String
    cityStateZip: java.lang.String  autoForm1.dataOutput.postalcode
    urbanization: java.lang.String
    casing: strikeiron.usaddressverification.CasingEnumType
    addressLine2: java.lang.String
    addressLine1: java.lang.String  autoForm1.dataOutput.addressline1
  licenseInfo: strikeiron.usaddressverification.LicenseInfo
```

At this point, we could go through and bind more input parameters to more fields. However, for this example, we're going to keep it simple.

14. We do have to set up a binding for the **licenceinfo** input. Under "Bind Targets", click to expand **licenseinfo**.

15. Under **licenseinfo** click to expand **registeredUser.**

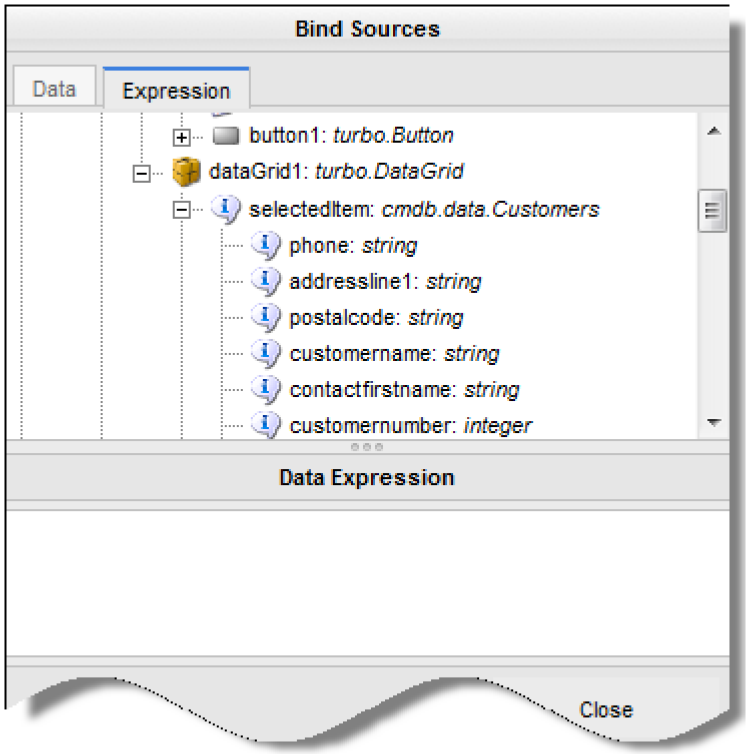
16. Under **registeredUser**, click to select **userID**.



Bind Targets
```
input: VerifyAddressUSAInputs
  parameters: strikeiron.usaddressverification.VerifyAddressUSA
    firm: java.lang.String
    cityStateZip: java.lang.String  autoForm1.dataOutput.postalcode
    urbanization: java.lang.String
    casing: strikeiron.usaddressverification.CasingEnumType
    addressLine2: java.lang.String
    addressLine1: java.lang.String  autoForm1.dataOutput.addressline1
  licenseInfo: strikeiron.usaddressverification.LicenseInfo
    registeredUser: strikeiron.usaddressverification.RegisteredUserType
      password: java.lang.String
      userID: java.lang.String  (expression)
    unregisteredUser: strikeiron.usad ification.UnregisteredUserT
```

17. Next, in the “Bind Sources” list, we select the source for the authentication string. In this case we have a string that you can use for authentication.

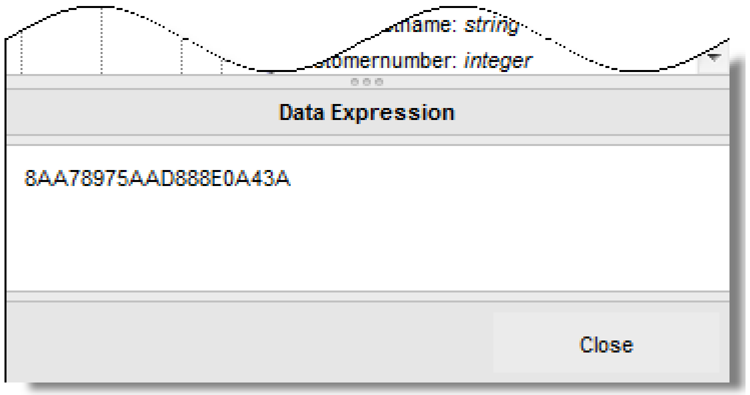
    So far in this tutorial we have used widget and Service Call values for our bind sources. However, you can also use data expressions for bind sources, which we need to do now. Under “Bind Sources” click the **Expression** tab.



All the bind sources available on the **Data** tab are also available on the **Expression** tab, but the **Expression** tab also includes an area where you can write a data expression. If you click a bind source in the tree, it is automatically represented in the “Data Expression” area below. However, in this example, we are simply going to use a constant.

18. In the “Data Expression” field, type in the following string:

    ```
    8AA78975AAD888E0A43A
    ```



8AA78975AAD888E0A43A

NOTE: We don't need to set a value for the password; this userID code is sufficient for this particular service.

19. Click **Close**.

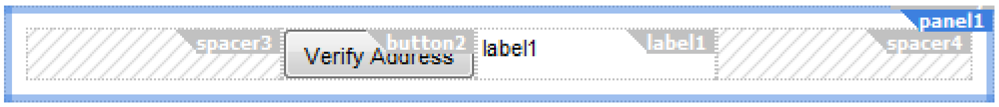
20. Save the project.

The next step is to "Add the Address Verification Button" on page 46.

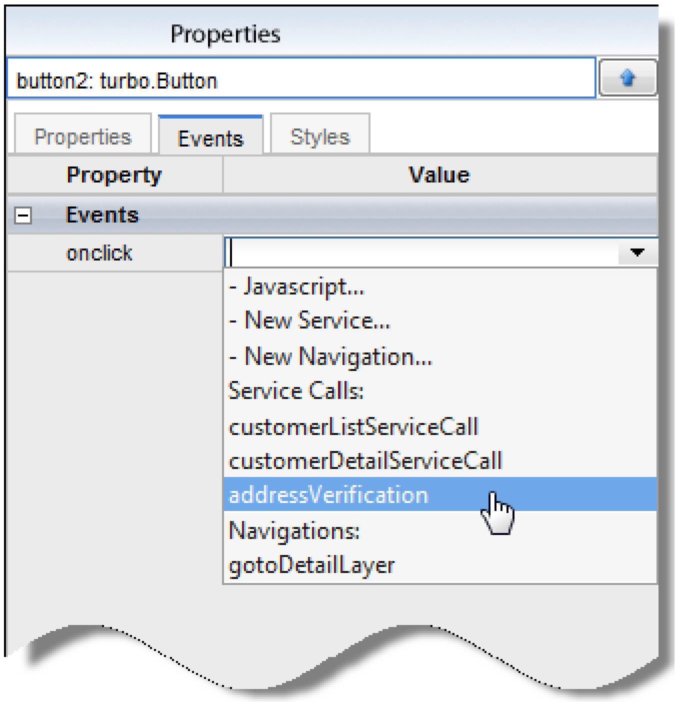
Add the Address Verification Button

In this section we add a Button widget to call the address verification service and a Label widget to display the results of that Service Call. Follow these steps:

1. In the **Widgets** tab of the Model Tree, find the DetailLayer.
2. From the Palette, drag a Panel widget into the DetailLayer, above the AutoForm widget.
3. Now we add the Button widget, the Label widget and some spacers to center the Button in the Panel. From the Palette, drag the following widgets into your new Panel widget:
   - ∞ a *Spacer* widget with the **sizeUnits** property set to **flex** and the **size** set to 1
   - ∞ a *Button* widget with the **caption** property set to: `Verify Address`
   - ∞ a *Label* widget
   - ∞ another *Spacer* widget with the **sizeUnits** property set to flex and the **size** property set to 1
4. Drag the widgets inside the panel until they are in the correct order.
5. Size the Panel down a little so that is about the right height for a button and a text input. The Panel should look something like the one shown below.

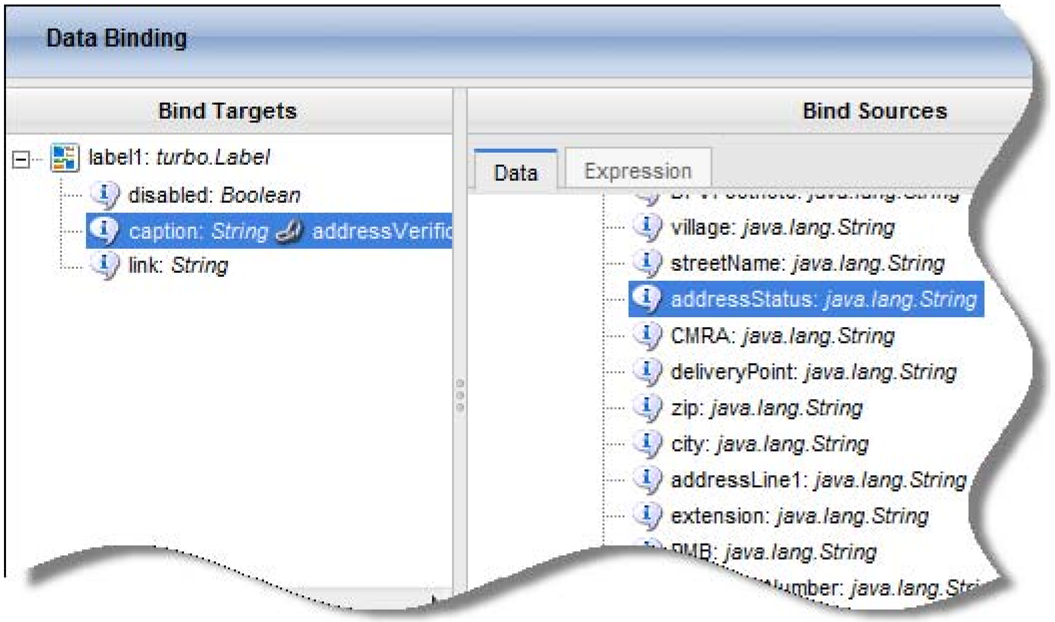


6. Next we set up the Button widget to call our address verification service.
7. Select the Button widget in the Page Designer.
8. In the Property Editor, click the **Events** tab.
9. Click the button next to the **onclick** property. A drop-down menu appears, listing the different ways you can wire up this button.
10. Under **Service Calls:** choose the **addressVerification** option.



11. Finally, we bind the Label widget to the output from the Service Call.
12. In the **Widgets** tab of the Model Tree, select the Label widget.
13. In the Property Editor, click the **Properties** tab.
14. Click the **binding** property button. The Data Binding dialog appears.
15. Under “Bind Targets”, select **caption**.
16. Under “Bind Sources”, scroll down to the bottom of the list and expand **addressVerification**.

17. Below that, click to expand **VerifyUSAResult**. Here, select the **addressStatus** string. When you make this selection, the link icon appears in the "Bind Targets" list, showing that the binding is complete.



18. Click **Close**.

19. Save the project.

20. Run the application. When you click on the Verify Address button, your application will use the Web Service to check whether the address is valid based on the postal code and addressline1. It will display a message that says either "Valid" or "Invalid", depending on the results.

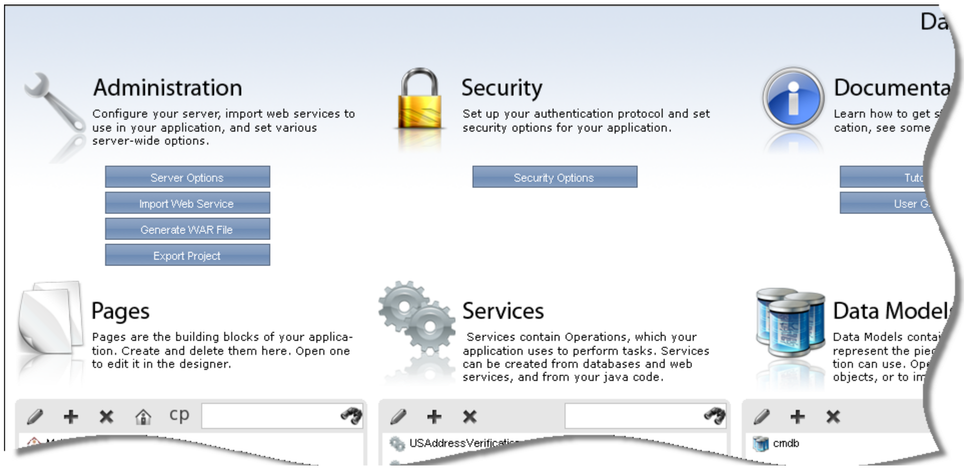
Congratulations!

You have added a Web Service Call to your application!

Glossary

Container. A container is a type of widget that is primarily designed to group and organize the widgets that it contains. The panel widget is a commonly-used type of container widget. A pane widget is a special container widget that you use to hold a sub-page.

Dashboard. The Dashboard screen in the WaveMaker Studio provides an overview of all the components in the currently-open project. To see the Dashboard, open the project in the WaveMaker Studio and then click the **Dashboard** tab (in the Editor tabs across the top left of the Studio).



Da

Administration

Configure your server, import web services to use in your application, and set various server-wide options.

Security

Set up your authentication protocol and set security options for your application.

Documenta

Learn how to get s
cation, see some

Pages

Pages are the building blocks of your application. Create and delete them here. Open one to edit it in the designer.

Services

Services contain Operations, which your application uses to perform tasks. Services can be created from databases and web services, and from your java code.

Data Model

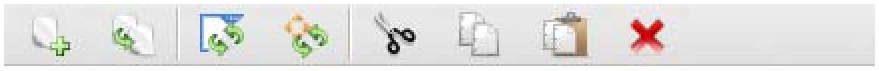
Data Models contai
represent the pie
tion can use. Op
objects, or to im

Database Service. The service that WaveMaker Studio creates for you when you import a database. Each database service contains predefined operations (insert, update, delete, and so on) for interacting with the tables in that database.

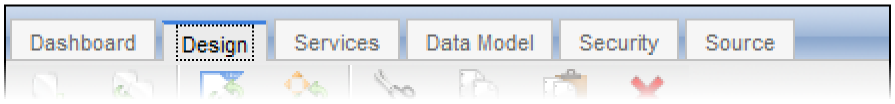
Data Binding. A data binding is the way you get application data to or from a widget. You specify the data binding by defining the **binding** property for a widget. You can bind to service calls, database columns, variables and so on.

Data Model. A data model describes a set of database tables that you want to be able to access in WaveMaker. The easiest way to create a data model is to import existing data tables by going to the **Data Model** tab in WaveMaker, selecting the **Import** Data sub-tab, and entering the database login information.

Design Toolbar. The main toolbar in the Page Designer.



Editor Tabs. The Editor tabs allow you to access the different data, service and design components of your application. When a project is open in WaveMaker Studio, these tabs are visible across the top left of the screen.



Export Project. WaveMaker Studio can export projects in zip format to facilitate sharing of projects through email. You export projects from the Dashboard by selecting **Export Project** under the “Administration” heading.

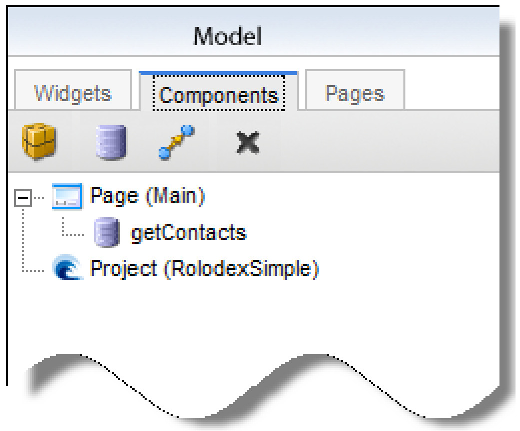
Layers. Layers provide an alternative to pages in WaveMaker Studio. To set up layers, you use a special widget called the Layers widget. Add one or more Layer widgets to the Layers container. You typically set up navigation between the Layer widgets by using accordion groups or tabs. Alternatively, you can set up a Navigation component to reveal a different Layer.

LayoutBox. The LayoutBox is the top container widget in the Page Builder. This widget is used by the WaveMaker Studio only. You cannot delete it or add a new one.

LiveLayout. The LiveLayout™ feature allows you to see live data in the Page Designer as you are working with your pages. This allows you to dynamically test run pieces of the application as you assemble it, rather than having to run the whole application.

Main Page. Automatically-generated page that serves as the default page to be loaded for the application. You are free to change the default page to a different page and you are free to delete the “Main” page from your project.

Model Tree. The Model Tree is located on the upper right side of the Page Designer. The Model Tree has three tabs: The Widgets tab shows a hierarchical tree of all the widgets in the current page; The Components tab shows the Service Calls, Navigation components, and Variables available to your page; and the Pages tab shows all the pages in the current project.



Navigation Component. An application component that loads a particular Layer or Page. In the case of pages, the Navigation component also stores information about where to display the page. To create a Navigation component, open the Components tab in the Model Tree and click the new Navigation component icon.

Operation

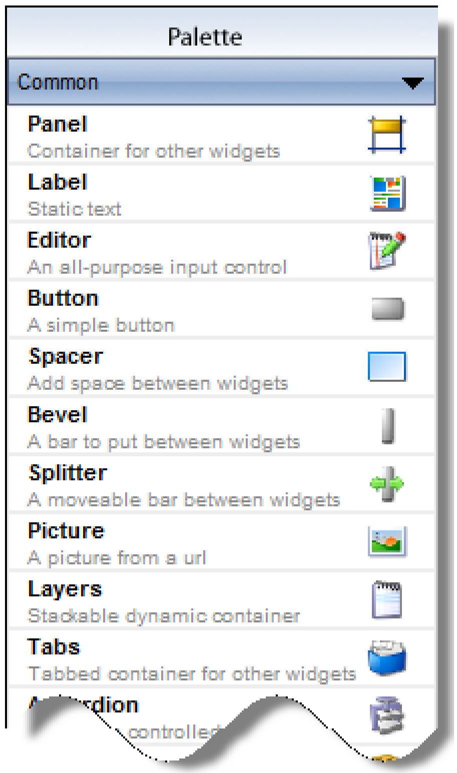
Page. A unit of application development in studio. Once you save a page, it appears in the Pages section of the Palette and you are free to use it as a subpage. To set up navigation between pages, use a Navigation component. You can use Layers as an alternative to pages.

Page Builder. The area in the Page Designer where you arrange your widgets. Drag widgets from the Palette and drop them into the Page Builder. The Page Builder area is contained by a special widget called the layoutBox. The layoutBox cannot be moved or deleted.

Page Container. A pane widget.

Page Designer. An Editor in the WaveMaker Studio that allows you to edit your application's pages, service components and variables. To open the Page Editor, click the Design tab in the Editor tabs across the top of the WaveMaker Studio.

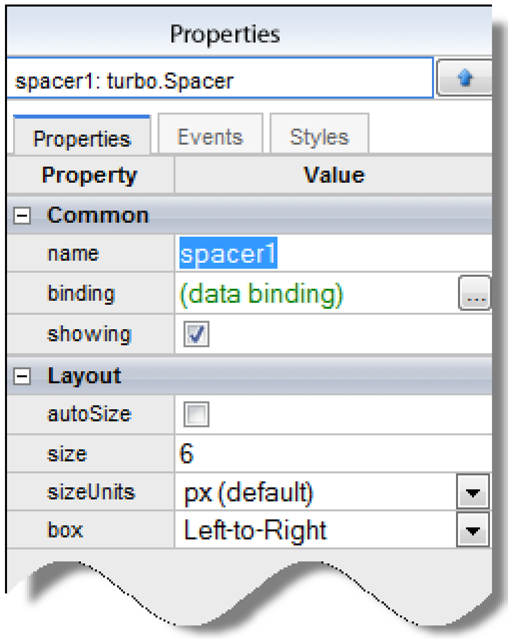
Palette. The Palette is located on the left of the Page Designer. It contains all the widgets that are available to your application. Drag widgets from the Palette into the Page Builder.


Palette
Common
Panel
Container for other widgets
Label
Static text
Editor
An all-purpose input control
Button
A simple button
Spacer
Add space between widgets
Bevel
A bar to put between widgets
Splitter
A moveable bar between widgets
Picture
A picture from a url
Layers
Stackable dynamic container
Tabs
Tabbed container for other widgets

Pane. A container for showing a page as a sub-page. WaveMaker Studio automatically creates a pane widget for you when you create a sub-page by dragging an existing page from the Pages section of the Widget Panel into the Page Designer. You can also drag a Pane widget onto any page and use it to dynamically display different pages as subpages.

Panel. A panel widget is a container for other widgets like calendars and editors. A panel is usually invisible to the end user, it is primarily used as a useful way to group a set of widgets together. The contents of the Panel widget are arranged either from top to bottom or from left to right, depending on the value of the **box** property.

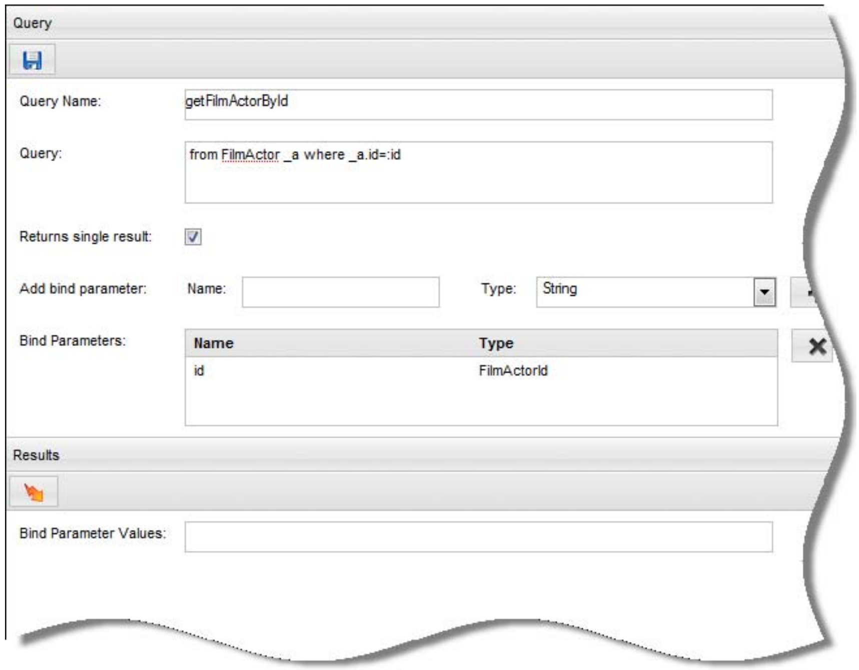
Property Editor. Panel in the WaveMaker Studio that allows you to edit the properties for whatever widget or component is selected in the Page Builder. Items that have properties include widgets, service calls, and variables.



spacer1: turbo.Spacer

Property	Value
Common	
name	spacer1
binding	(data binding)
showing	☑
Layout	
autoSize	☐
size	6
sizeUnits	px (default)
box	Left-to-Right

Query Editor. The area of the Data Model editor that allows you to view, create and test database queries. To see the Query Editor, select an existing query in the Data Objects list in the Data Model Editor.



Query Name: getFilmActorById

Query: from FilmActor _a where _a.id=:id

Returns single result: ☑

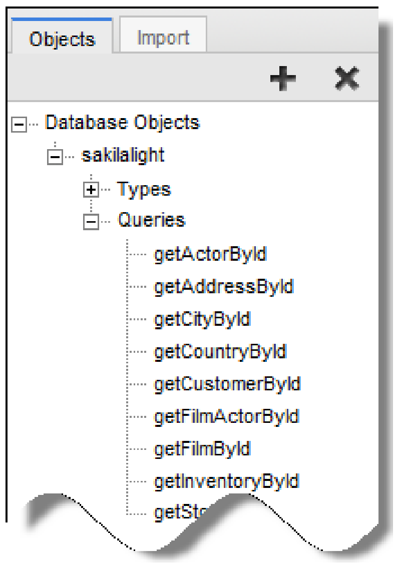
Add bind parameter: Name: Type: String

Bind Parameters:

Name	Type
id	FilmActorId

Bind Parameter Values:

Queries. .Database queries. When you import a database, WaveMaker Studio automatically generates a few basic database queries for you. These are listed in the Data Model Editor under Data Objects. You can use these generated queries as they are, modify them to suit your needs, or create new queries from scratch.



Remote Services. Remote services are services defined outside the WaveMaker Application. These include SOAP and REST services, and RSS/Atom feeds.

Service. A service is a group of one or more operations that your application can call. For example, a database service provides a variety of operations that allow you to operation on the database (such as update, delete, or insert). Each service has to be added to the project in order to access it in that application. Once you've added a service to your project, you can create Service Calls to call operations in that service..

Service Call. A Service Call is a way for a WaveMaker application to invoke an action, such as a database query, navigation from one page to another or a custom Java method. To create a Service Call component, open the Components tab in the Model Tree and click the new Service Call component icon.

Services Editor. The Services editor displays all the services that are registered for the current project. You can use the **Add** tab to add a new Java or Web service. Database services are automatically created when you import a database. To access the Service editor, click the **Services** tab.

Type. In the Data Model, the tables in each database are represented as Types (these are actually Java types used to interact with the database, but you do not need to worry about that.) These Types are listed in the Data Model Editor under Data Objects.

Variable. A component that you can create to hold data that you can bind to service calls or widgets. To create a Variable component, open the Components tab in the Model Tree and click the new Variable component icon.

Web Services. Application services that an application can access from a Web server. These include SOAP services, REST services, and RSS or Atom feeds.

Widget. A widget is a visual component that can be used to create a web application. Examples include calendars, text editors, pictures and containers. To add a widget to a page, drag it from the Palette and drop it in the Page Builder. To configure the widget, modify its properties in the Properties Editor.

WaveMaker