Sencha

Getting Started with Designer

Sencha, Inc.

www.sencha.com
525 University Avenue, Suite 23
Palo Alto, CA 94301

Contents

What is Sencha Designer?

Designer is a graphical user interface builder for Ext JS Web applications. Its easy-to-use drag-and-drop environment enables you to quickly prototype your application's interface components, connect the interface components to your data, and export well-formed, object-oriented code for each component.

Programmers and non-programmers alike can use Designer to collaborate on an application's design, which helps get projects started faster and enables faster iteration. With Designer, you can:

- Quickly and easily build complex forms.
- Change component layouts and swap control types with the click of a button.
- Focus on writing implementation code, rather than boilerplate UI code.

Additional Information

For more information about Designer and Ext JS:

- Watch the Designer Demo for a quick introduction to Designer.
- For information about the latest Designer release and updates, see the Designer Changelog.
- If you're new to Ext JS, see the component and container model information in the Ext JS Overview.
- For the details about any Ext JS class or method, see the Ext JS API Reference.

Building a Web App UI with Designer

Designer is designed to be used in conjunction with your existing development environment and tools. It's not a replacement for Eclipse or your favorite text editor. The code generated by Designer can be imported into your existing IDE, and you can edit the UI implementation files outside of Designer from within your IDE or with the editor of your choice.

When using Designer, you:

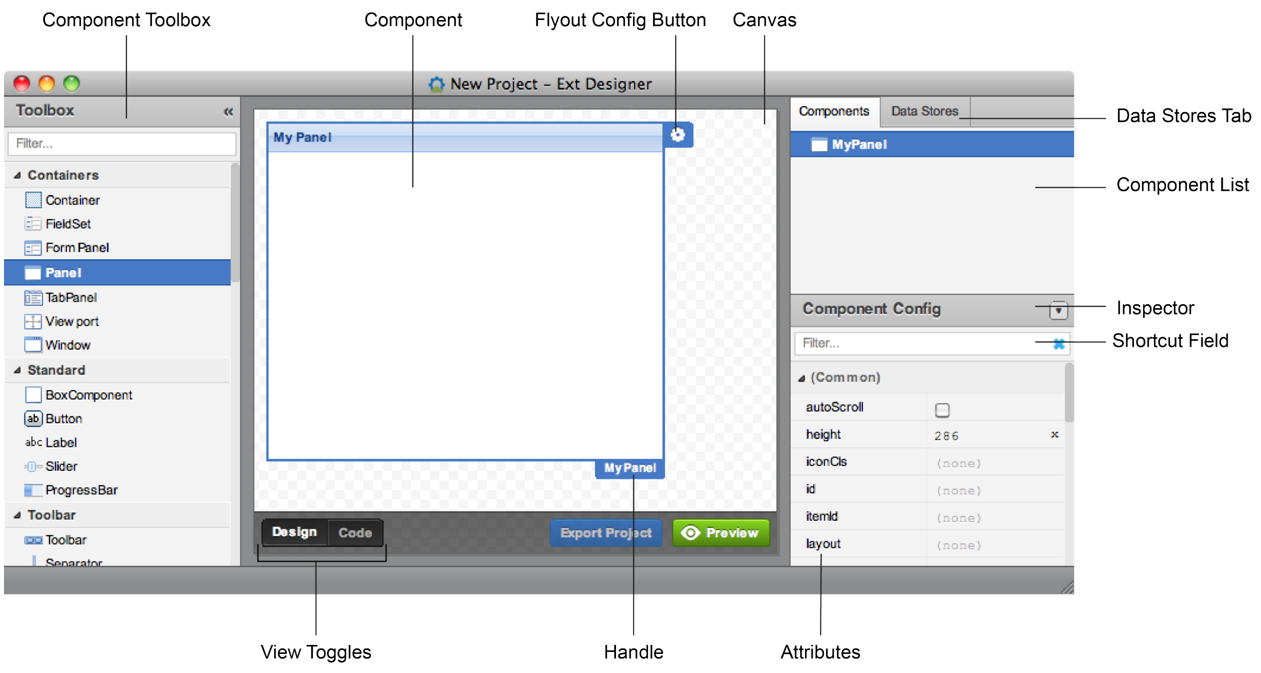
1. Lay out your UI components on the Designer canvas.
2. Configure the components.
3. Connect to your data stores.
4. Export your project.
5. Implement your event handling and custom methods in the generated .js file.

Important! When you export your project for the first time, two Javascript files are created for each top-level component in your project. The file with the .ui.js extension contains the base class for the UI component. You extend this base class in the file with the .js extension to implement your event handlers and custom functions. DO NOT modify the .ui.js file directly, it will be overwritten whenever you modify and export your project.

You can iterate through this process until you are satisfied with your UI. As long as you only make changes to the .js file, you can regenerate the UI code as many times as you want.

Navigating Designer

When you launch Designer, it automatically displays a new project with a blank canvas.



- Toolbox—contains all of the components you can use to build your UI. These correspond to standard Ext JS classes. For more information about each class, see the Ext JS API Reference. You can drag and drop components from the toolbox onto the canvas.
- Canvas—provides a design space for you to assemble your UI. You can resize and rearrange components you've added to the canvas and edit component titles and labels. (You can only reposition absolute-positioned components, for more information see Laying Out UI Components with Designer.)
- Components—lists all of the components you've added to your project. From the Components tab, you can select, rearrange, duplicate, transform, and delete components you've added to the canvas. You can view and modify the selected component's settings in the Component Config pane.
- Data Stores—shows the data sources you've added to your project. From the Data Stores tab, you can add new JSON, Array, XML, and Direct data sources, add and remove a source's data fields, and select, duplicate, or delete existing sources. You can view and modify the selected data store's settings in the Component Config pane.
- Component Config—view and modify settings for the selected component or data store.

As you add components to the canvas, you can see what they will look like in a web browser by clicking the Preview button below the canvas. You can see what the generated Javascript code looks like by toggling between the Design and Code views. You save the generated code to an external file by clicking the Export button. Note that you'll need to save your new project before you can export it.

Shortcuts

Designer provides a number of navigation and configuration shortcuts. You can:

- Double-click components in the Toolbox to add them to the canvas.
- Tab between inline editable fields on the canvas.
- Locate particular attributes in the Component Config inspector by typing in the Shortcut Field.
- Set attribute values using the Shortcut Field by entering the name of the attribute followed by a colon and the value you want to set. For example, *title: Car Listings*.

Anatomy of an Designer UI

When you lay out UI components with Designer, you drag a container such as a Window or FormPanel onto the canvas and add components to the container. By adding additional top-level containers to your project, you can lay out the different parts of your UI as separate entities. When you export your project, each top-level container is represented by a class with the code for that class in a separate file. This gives you flexibility in how you assemble the elements on your web pages, makes it possible to reuse common components, and makes it easier to maintain your implementation code.

Laying Out UI Components with Designer

Designer leverages the powerful layout capabilities of Ext JS to simplify the creation of complex forms and make it easy to switch between alternate layout options.

Layout Options

When you set the layout on a container, it controls how Ext JS lays out the components within that container. You can switch between layout options by clicking a Container's flyout config button and selecting a different layout.

ExtJS provides a number of basic container layouts. Some support specific, commonly-used presentation models such as accordions and cards, while others provide more general-purpose models that can be used for a variety of applications.

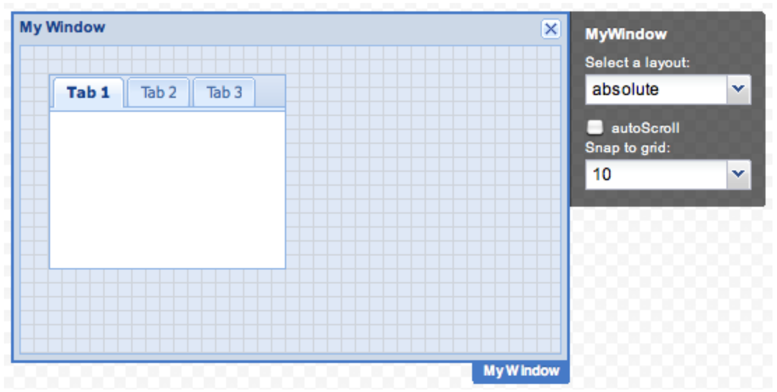
auto

If no other layout is set for a container, it defaults to the auto layout. For general purpose containers such as a Panel, this means child components will be rendered sequentially. Note that some containers are automatically configured to use a particular layout. For example, FormPanel defaults to the form layout and TabPanel defaults to the card layout.

absolute

Arranges components using explicit x/y positions relative to the container. This enables you to explicitly reposition and resize components within the container. While this gives you fine-grained control over the layout, keep in mind that absolute-positioned components remain fixed even if their parent container is resized.

When you use the absolute layout, Ext Designer displays a grid within the container. By default, components are snapped to the grid as you reposition them. You can change the grid size or disable the grid by clicking on the container's flyout config button. The grid is only displayed as a layout guide in the Design view, it is not visible when the component is rendered.



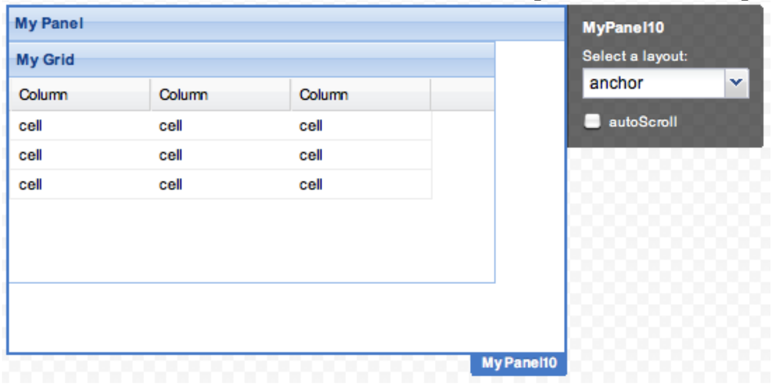
MyWindow
Select a layout: absolute
autoScroll
Snap to grid: 10

accordion

Arranges panel components in a vertical stack where only one panel is expanded at a time. Only panels (including FormPanel and TabPanel) can be added to a container that uses the accordion layout.

anchor

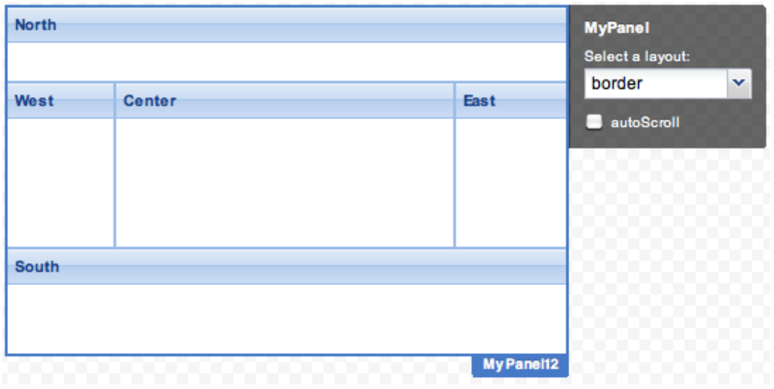
Arranges components relative to the sides of the container. You can specify the width and height of child components as a percentage of the container or specify offsets from the right and bottom edges of the container. If the container is resized, the relative percentages or offsets are maintained.



border

Arranges panel components in a multi-pane layout. Panels are arranged in the container by assigning them to one of five regions: North, South, East, West, or Center. A container that uses the border layout has to have a child assigned to the Center region. The center panel is automatically sized to fit the available space. You can resize the North, South, East, and West panes on the canvas by clicking and dragging the right or bottom edge of the panel.

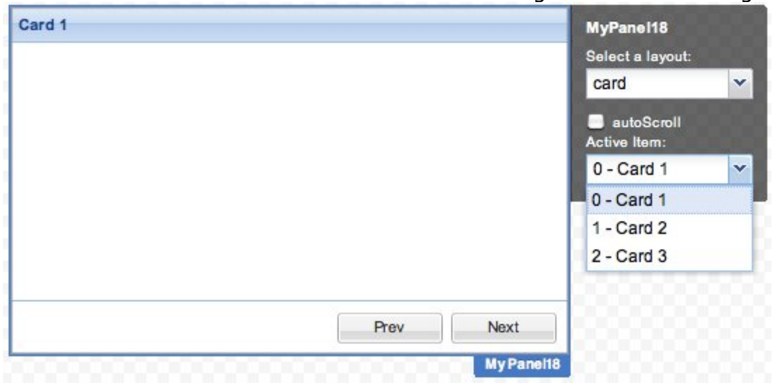
You can make any of the panels in a border layout collapsible by enabling the collapsible attribute. When rendered, the child panels automatically resize when the container is resized.



card

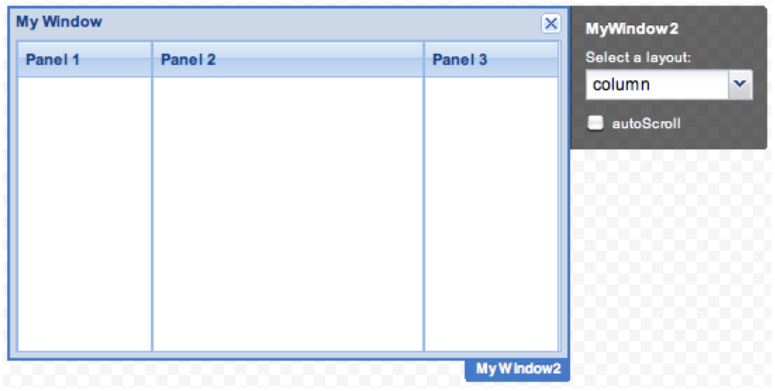
Arranges multiple child components but only one component is visible at a time. This layout can be used to step through a series of components and is commonly used to create wizards.

To specify the component that you want to make visible, you need to call setActiveItem. Typically, you attach this behavior to a navigation UI such as Previous and Next buttons in the footer of the container.



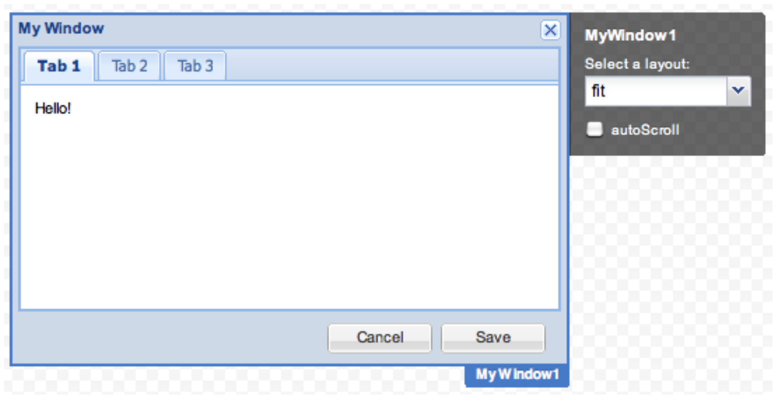
column

Arranges components in a multi-column layout. The width of each column can be specified either as a percentage (column width) or an absolute pixel width (width). The column height varies based on the contents. You can enable autoscroll so it's possible to scroll to view column contents that exceed the container height.



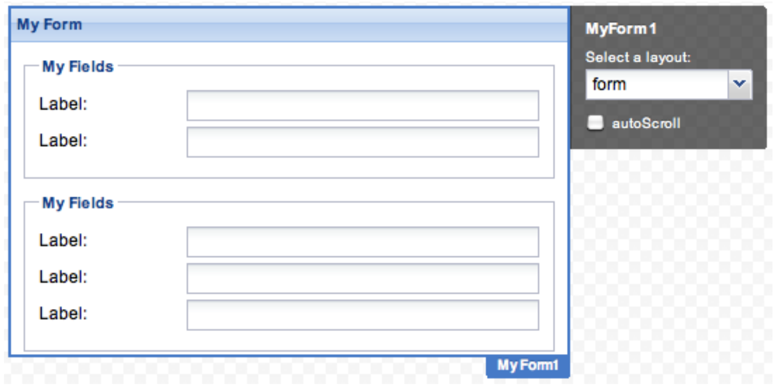
fit

Expands a single child component to fill the available space. For example, you might use this to create a dialog box that contains a single TabPanel. If the container is a type of panel component, you can also add a Toolbar to it.



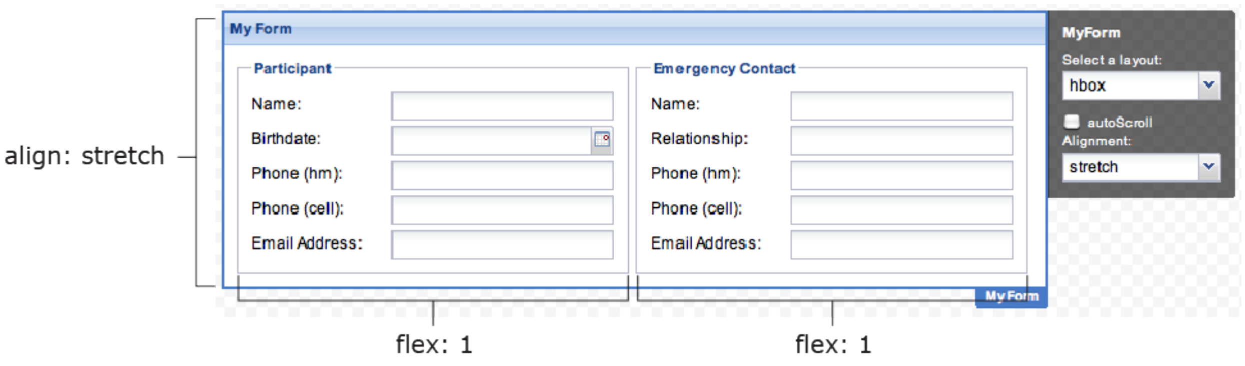
form

Arranges a collection of labelled form fields. A FormPanel uses the form layout by default.



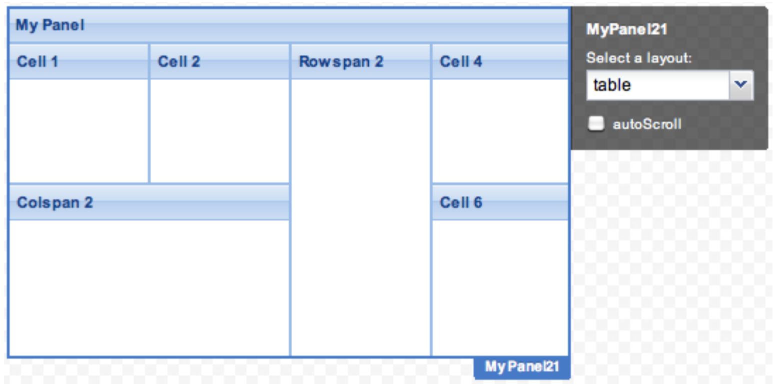
hbox

Arranges the child components horizontally. Setting the alignment of the container to stretch causes the child components to fill the available vertical space. Setting the flex attribute of the child components controls the proportion of the horizontal space each component fills.



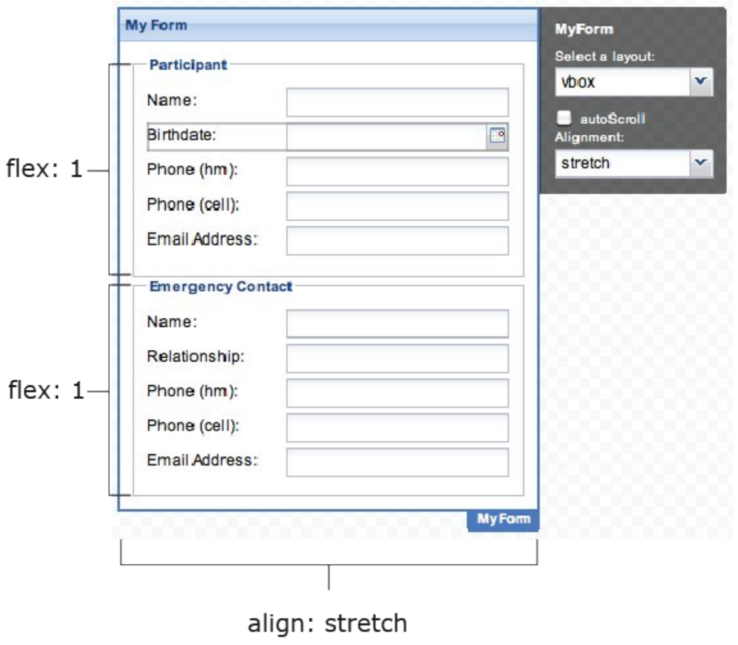
table

Arranges components in an HTML table. You specify the number of columns in the table and can create complex layouts by specifying the rowspan and colspan attributes on the child components.



vbox

Arranges the child components vertically. Setting the alignment of the container to stretch causes the child components to fill the available horizontal space. Setting the flex attribute of the child components controls the proportion of the vertical space each component fills.

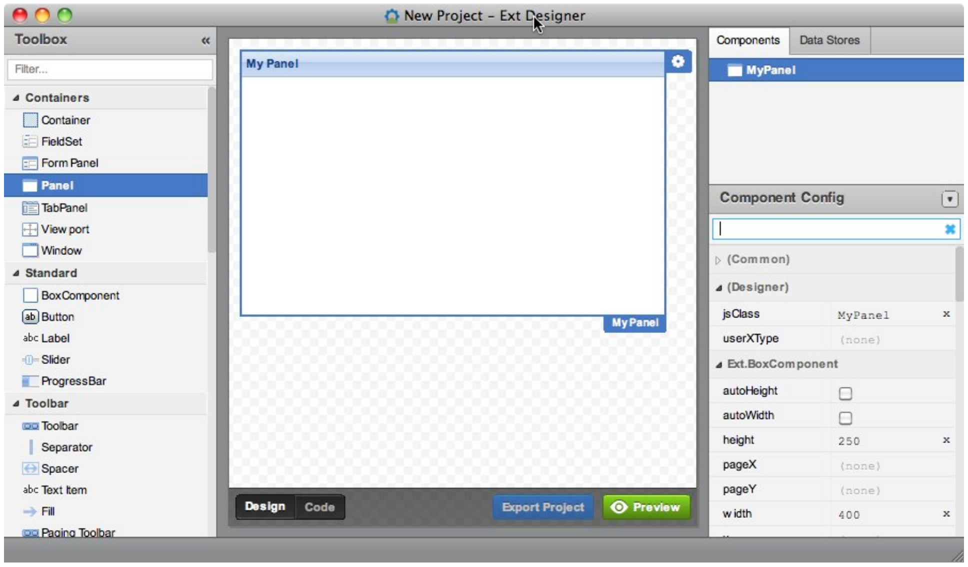


Adding Components

To assemble your application's UI, you drag components from the Toolbox onto the canvas. Designer ensures components are nested properly and won't let you add incompatible components to a container. For example, you can't drop a Window or Viewport component into a Container.

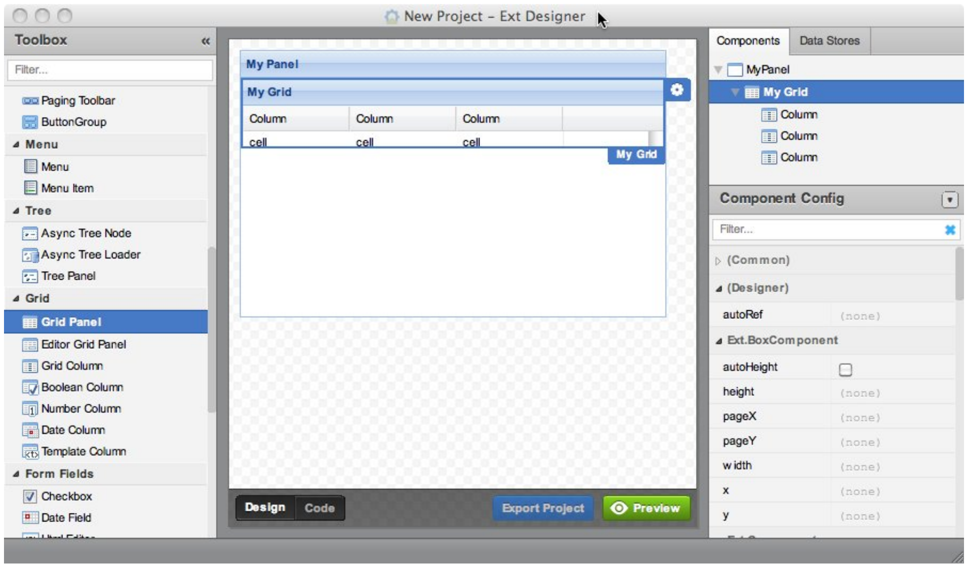
For example, to assemble the Car Listings UI shown in the Sencha Designer demo:

1. Drag a Panel container onto the canvas. This is the top-level component for the Car Listing application.



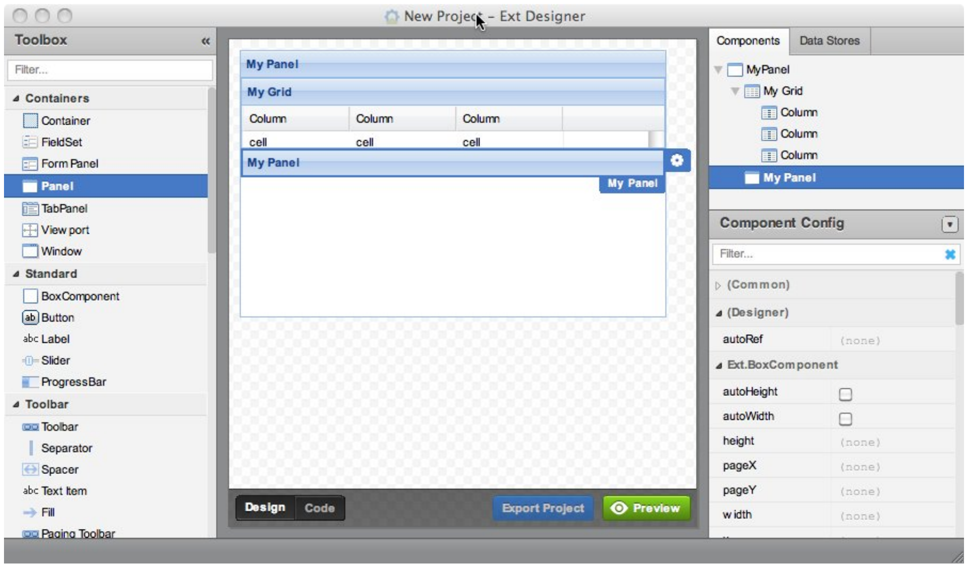
jsClass	MyPanel
userXType	(none)
autoHeight	☐
autoWidth	☐
height	250
pageX	(none)
pageY	(none)
width	400

2. Drag a Grid Panel into the Panel container. The grid panel will display the available car listings and enable the user to select a listing to view.



autoRef	(none)
autoHeight	☐
height	(none)
pageX	(none)
pageY	(none)
width	(none)
x	(none)
y	(none)

3. Drag another Panel into the Panel container. This panel will display the car details for the listing selected in the Grid Panel.



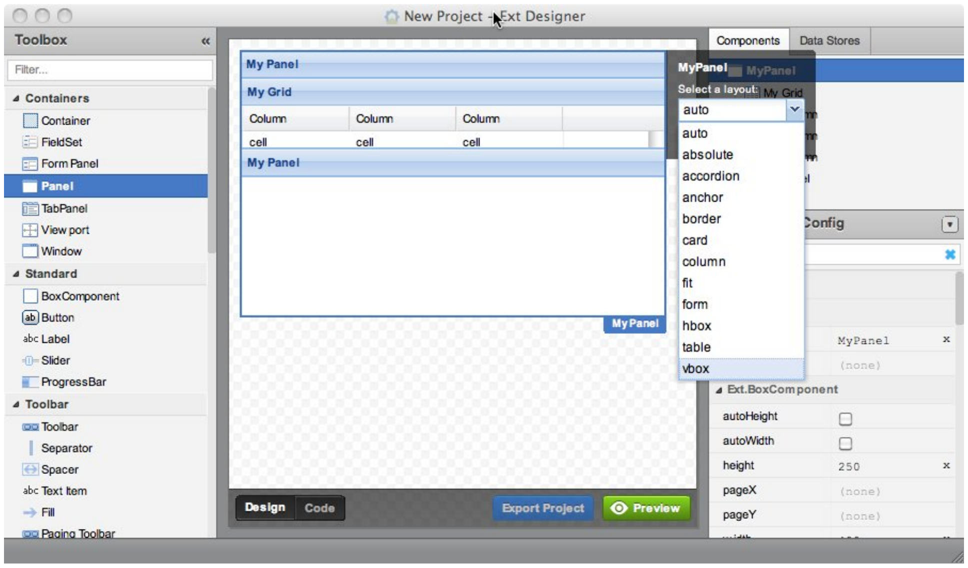
Positioning Components

By default, components are laid out using relative positioning. The best way to control the position of the elements on the canvas is to set the layout options on the containers and adjust the attributes that adjust the relative positions of each component.

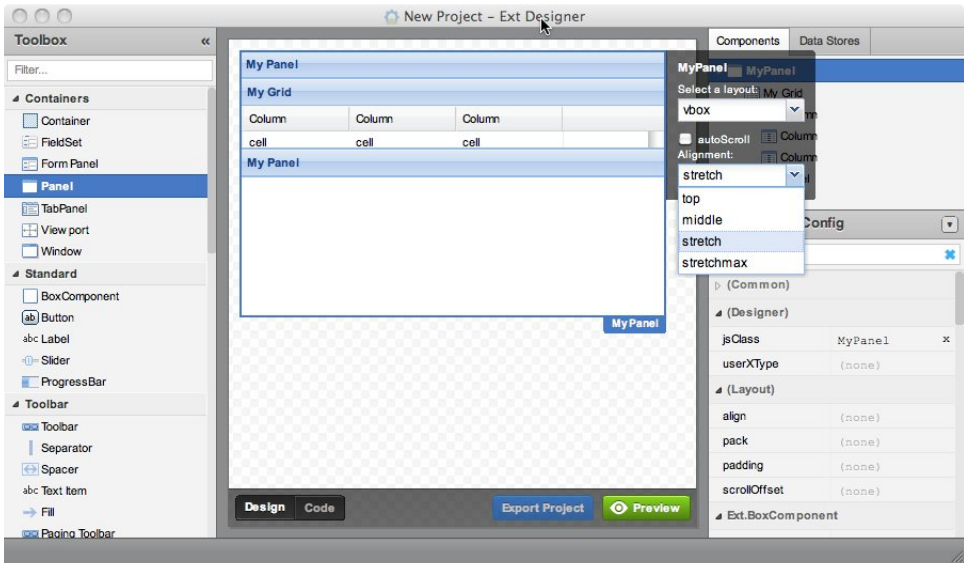
Note: If you choose the Absolute layout option, you can drag components around on the canvas to reposition them. (Generally, this isn't what you want to do. It's normally better to rely on the Ext JS layout manager to control the relative positions of the components.)

For example, to configure the layout of the components in the Car Listings UI:

1. Click the flyout config button on the top-level panel and set the layout to vbox. This will arrange the grid and subpanel vertically. From this menu, you can also set the alignment and auto-scroll attributes.



2. Set the alignment for the top-level panel to Stretch. This will cause the subcomponents to stretch to fill the available space.

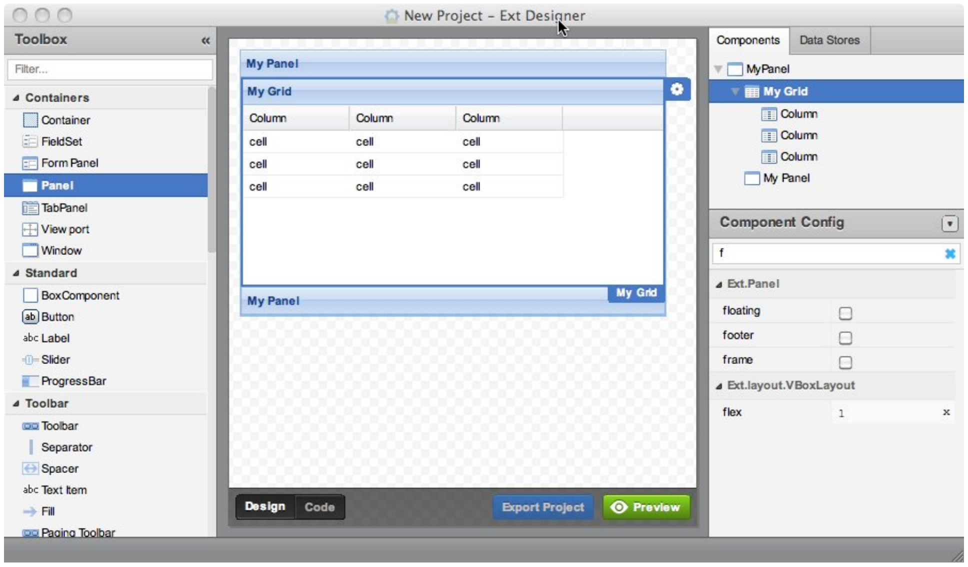


Select a layout: vbox
autoScroll
Alignment: stretch

jsClass	MyPanel
userXType	(none)
align	(none)
pack	(none)
padding	(none)
scrollOffset	(none)

3. Select the Grid Panel and set the flex attribute to 1 in the Component Config inspector.

   Tip: You can type the name or first few characters of an attribute in the text field at the top of the Component Config inspector to quickly navigate to a particular attribute.

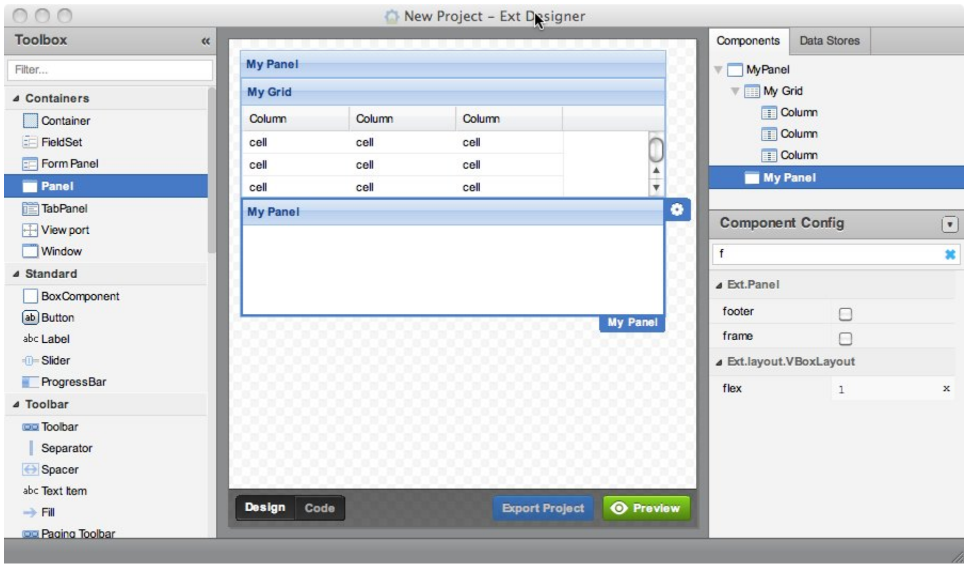


f

floating	☐
footer	☐
frame	☐
flex	1

The panel inherits the flex attribute from Ext.layout.VBoxLayout because the layout of the container is set to vbox. Setting the flex attribute of each of the components in the container to 1 will cause the components to take up the same amount of vertical space when the container is resized. (Similarly, if you wanted the subpanel to take up 2/3 of the vertical space, you could set the flex value of the panel to 2, and the flex of the grid to 1.)

4. Select the subpanel and set the flex attribute to 1.



f

Ext.Panel

footer	☐
frame	☐

Ext.layout.VBoxLayout

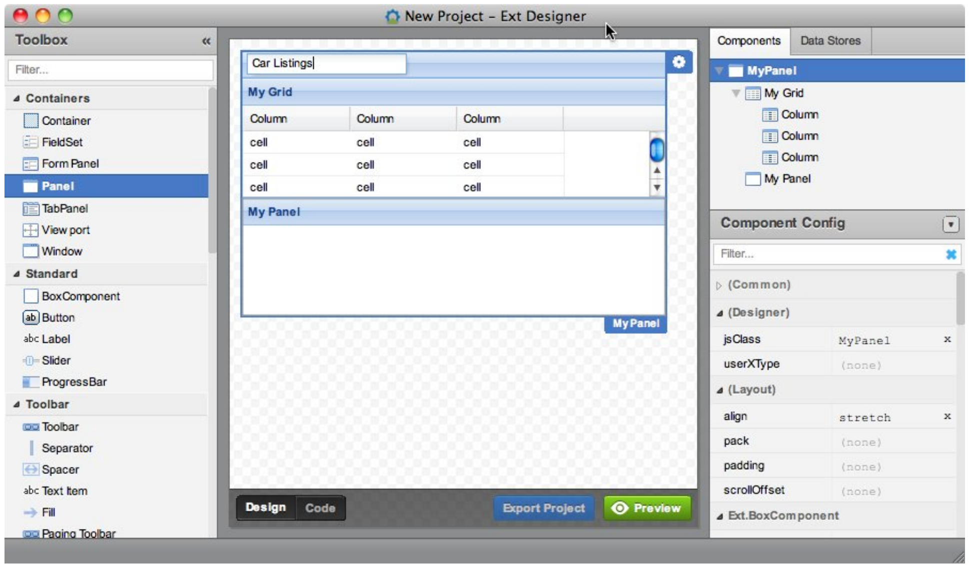
flex	1

Configuring Components

You can edit component attributes such as titles and labels directly. Simply double-click the text you want to modify and start typing. The Component Config inspector enables you to set all possible attributes for the selected component.

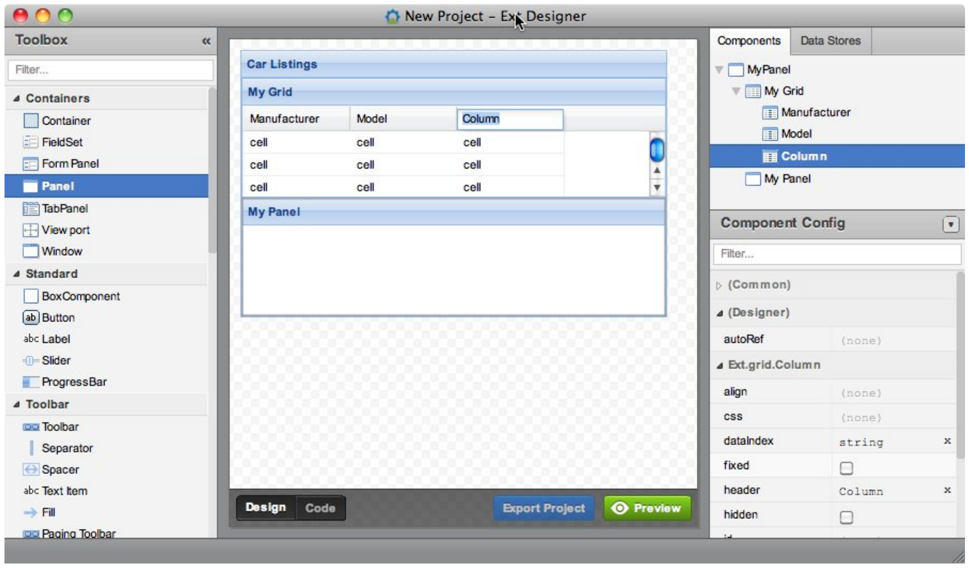
For example, you can directly set the title and column heading attributes for the Car Listings application:

1. Double-click the title bar of the top-level Panel to edit its title and change *My Panel* to *Car Listings*. This is the same as setting the title attribute in the Component Config inspector.



jsClass	MyPanel
userXType	(none)
align	stretch
pack	(none)
padding	(none)
scrollOffset	(none)

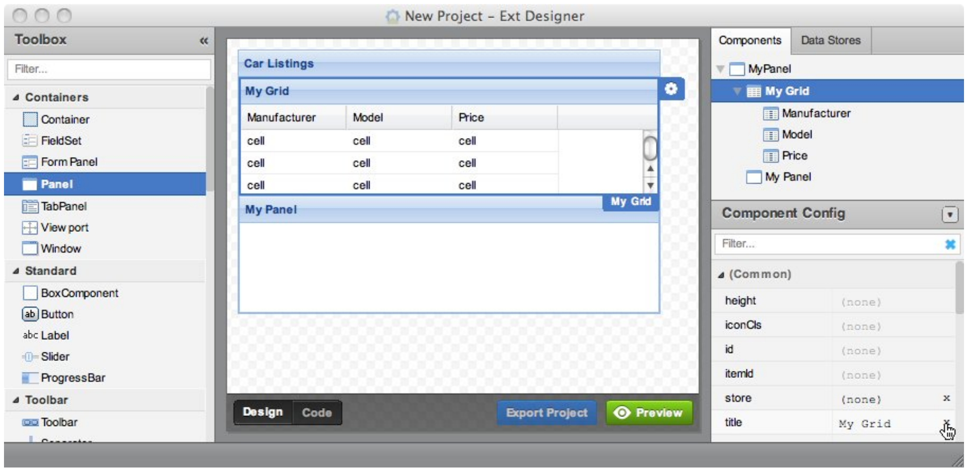
2. Double-click the column headings in the grid to set them to Manufacturer, Model, and Price. This is the same as setting the header attribute through the inspector.



autoRef	(none)
align	(none)
css	(none)
dataIndex	string
fixed	☐
header	Column
hidden	☐

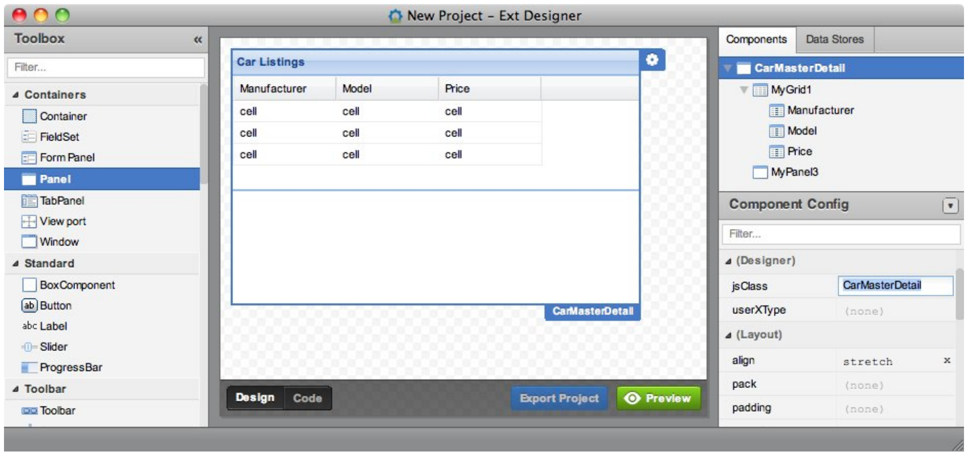
You can set the rest of the component attributes in the Component Config inspector:

1. Remove the title bars from the grid and subpanel by selecting each component and clicking the delete icon (x) to the right of the title attribute in the inspector. Now, the only title bar visible is the Car Listings title.



height	(none)
iconCls	(none)
id	(none)
itemId	(none)
store	(none)
title	My Grid

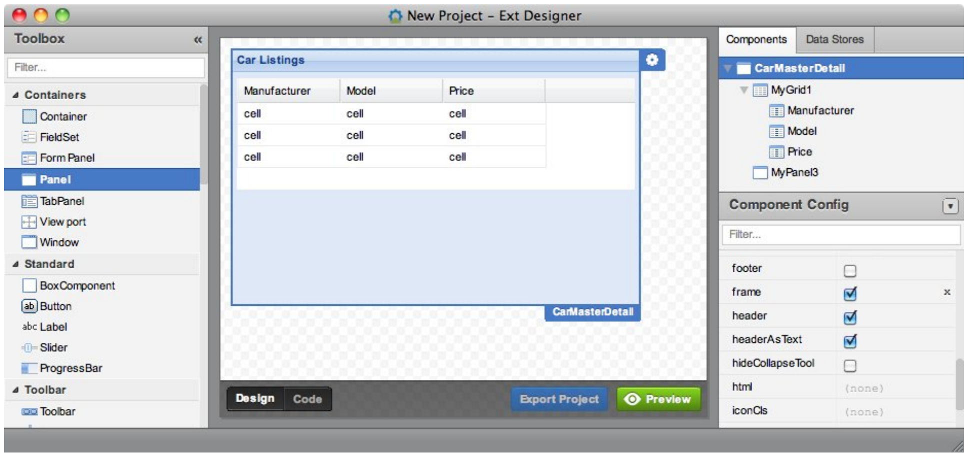
2. Select the Car Listings panel and set the jsClass attribute to CarMasterDetail. This will be the name of the component in the generated code.



jsClass	CarMasterDetail
userXType	(none)
align	stretch
pack	(none)
padding	(none)

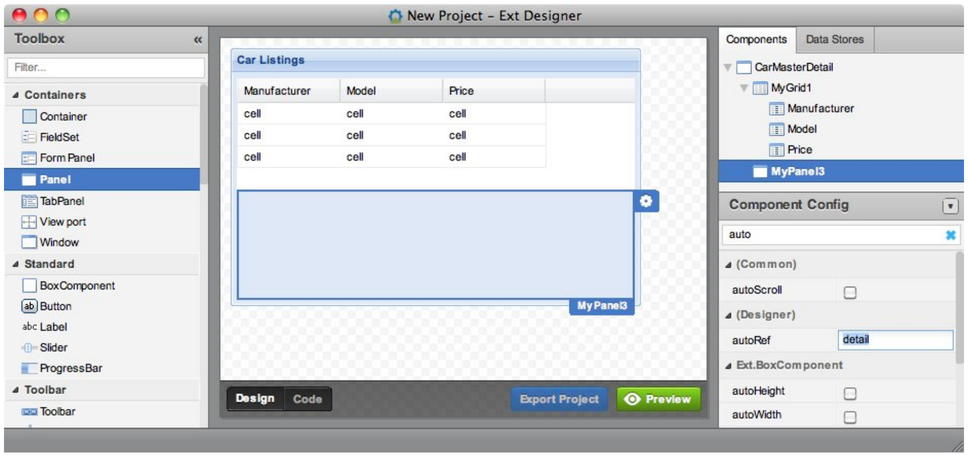
Note: You can toggle between the design and code views with the buttons below the canvas.

3. Enable the frame attribute of the Car Listings panel. Instead of the plain 1px square borders, this renders the panel with additional styling, including rounded corners.



footer	☐	
frame	✓	×
header	✓	
headerAsText	✓	
hideCollapseTool	☐	
html	(none)	
iconCls	(none)	

4. Configure auto references for the components so you can directly reference them in your code without worrying about how they are nested. Set the autoRef for the Grid Panel to *grid* and the autoRef for the subpanel to *detail*.



(Common)

autoScroll	☐

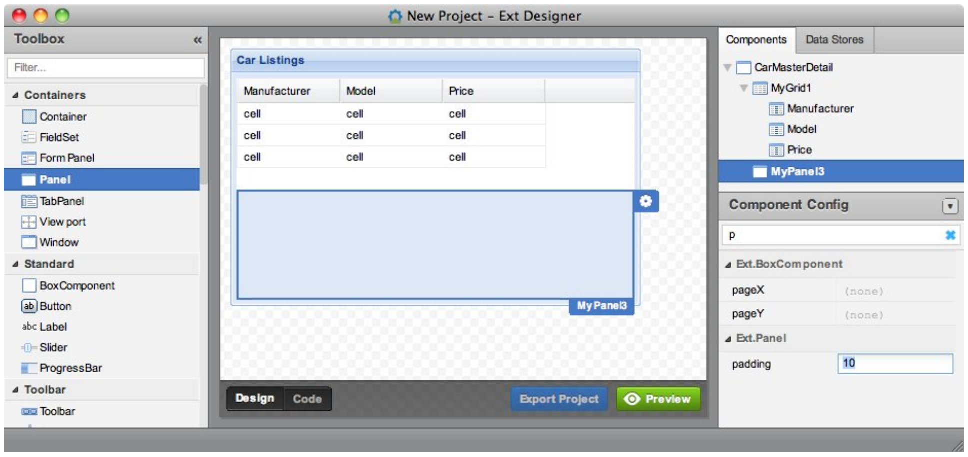
(Designer)

autoRef	detail

Ext.BoxComponent

autoHeight	☐
autoWidth	☐

5. To add some space padding around the contents of the subpanel, select the panel, type p to jump to the padding attribute, and set the value to 10. (This is the typical CSS padding attribute.)



Manufacturer	Model	Price
cell	cell	cell
cell	cell	cell
cell	cell	cell

```
p
Ext.BoxComponent
pageX      (none)
pageY      (none)
Ext.Panel
padding    10
```

Using Templates

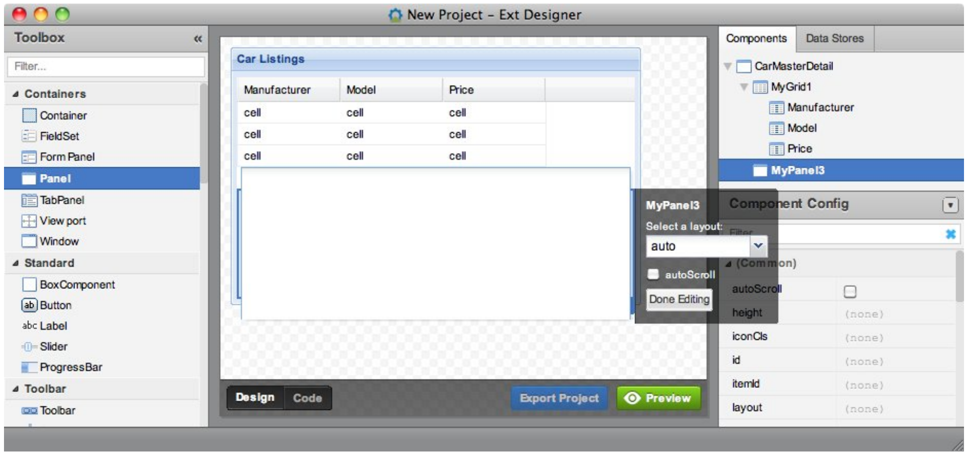
You can use templates to dynamically display information from a data store in a panel component. A template is an HTML fragment that can contain variables that reference fields in a data store. Templates also support auto-filling of arrays, conditional processing, math functions, and custom functions.

Variables are enclosed in curly braces. For example, `{manufacturer}` references the data field called *manufacturer*. You can also specify formatting functions to control how the data is displayed. For example, `{price:usMoney}` uses the usMoney format to prepend a dollar sign and format the number as dollars and cents. See Ext.util.Format for the full range of available formatting functions.

The Car Listings application uses a template to display the detail information for the selected listing. The image and wiki URL are pulled in from data fields in the cars.json data store. (See Connecting to Data for information about how to attach a data store.)

To configure the template:

1. Click the flyout config button on the subpanel and then click Edit Template to add a template for the detail information. The body of the component becomes an editable text area.



Manufacturer	Model	Price
cell	cell	cell
cell	cell	cell
cell	cell	cell

```
(Common)
autoScroll
height     (none)
iconCls    (none)
id         (none)
itemId     (none)
layout     (none)
```

2. Enter the HTML mark-up for the template:

```
<img src="cars/{img}" style="float: right" />
Manufacturer: {manufacturer}<br/>
Model: <a href="{wiki}" target="_blank">{model}</a><br/>
Price: {price:usMoney}<br/>
```

3. When you're finished editing the template, click **Done Editing**.

Connecting to Data

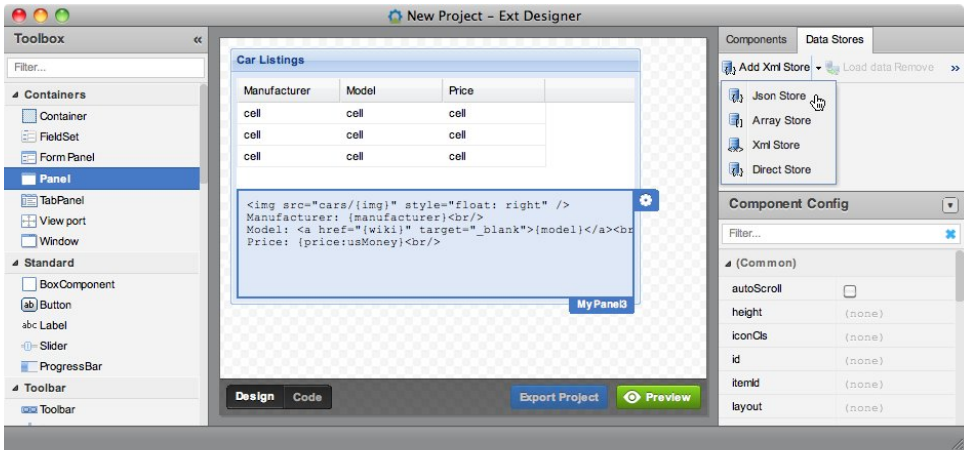
You can attach data stores and bind them to the components in your UI from within Designer.

For example, the listing information displayed by the Car Listings application is read from a JSON data store called cars.json. To connect the data store and pull in the manufacturer, model, price, wiki, and image data:

1. Add a data store for the cars data:

   a. Select the Data Stores tab.

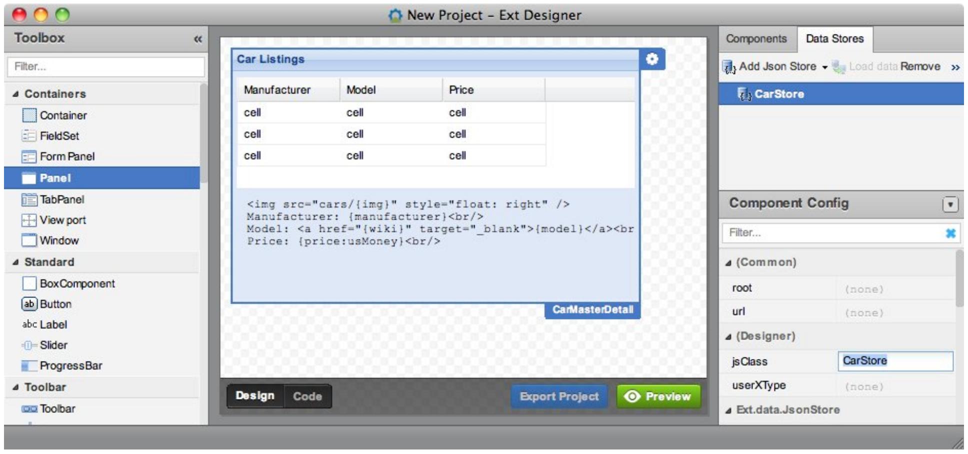
   b. Select Add Json Store from the Data Stores toolbar.



```
<img src="cars/{img}" style="float: right" />
Manufacturer: {manufacturer}<br/>
Model: <a href="{wiki}" target="_blank">{model}</a><br
Price: {price:usMoney}<br/>
```

autoScroll	☐
height	(none)
iconCls	(none)
id	(none)
itemId	(none)
layout	(none)

   c. Select the newly-created store and set the jsClass attribute to name the store CarStore.

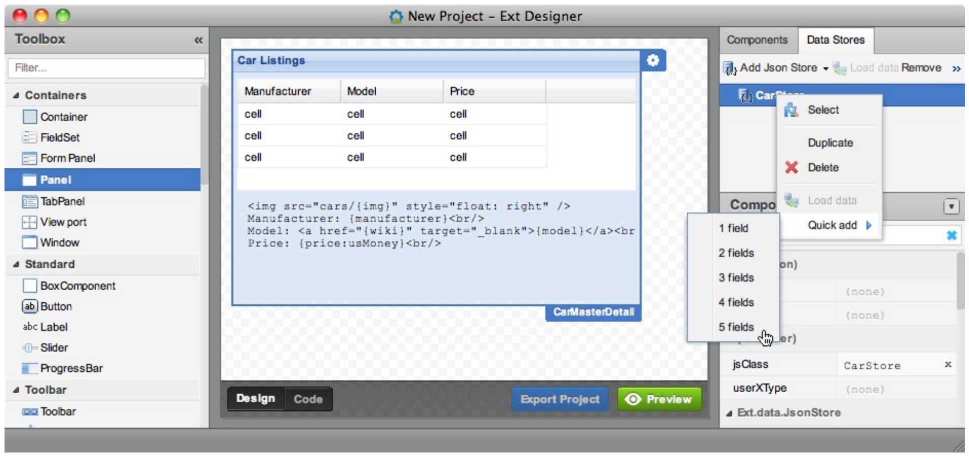


```
<img src="cars/{img}" style="float: right" />
Manufacturer: {manufacturer}<br/>
Model: <a href="{wiki}" target="_blank">{model}</a><br
Price: {price:usMoney}<br/>
```

root	(none)
url	(none)
jsClass	CarStore
userXType	(none)

   d. Set the storeId attribute to the same name. (The storeId is the name Designer displays in the list of available stores.)

2. Right-click the data store and select **Add Fields > 5 fields** to add data fields to the CarStore for each field defined in cars.json.

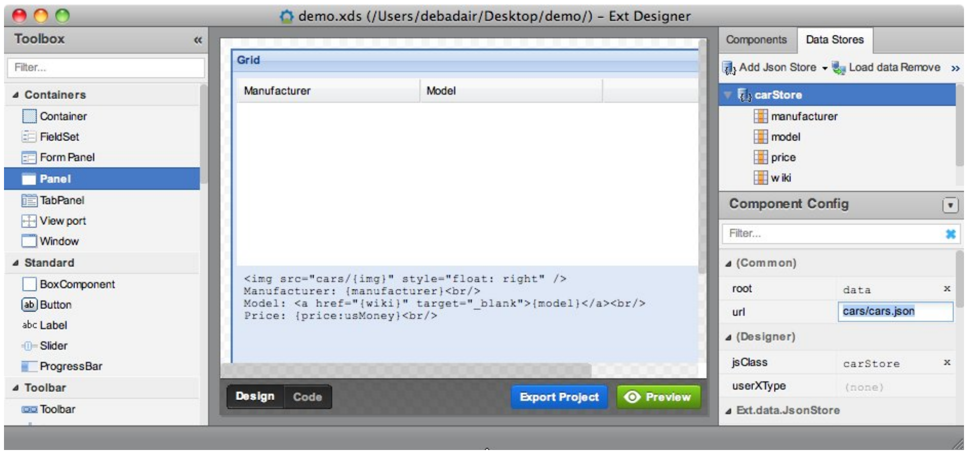


```
<img src="cars/{img}" style="float: right" />
Manufacturer: {manufacturer}<br/>
Model: <a href="{wiki}" target="_blank">{model}</a><br
Price: {price:usMoney}<br/>
```

jsClass	CarStore
userXType	(none)

3. Configure the CarStore:

   a. Set the url attribute to the relative path where the store will reside, cars/cars.json. This path is relative to the URL prefix specified in the Project Settings. To change the URL prefix, select Edit Preferences from the Edit Menu.



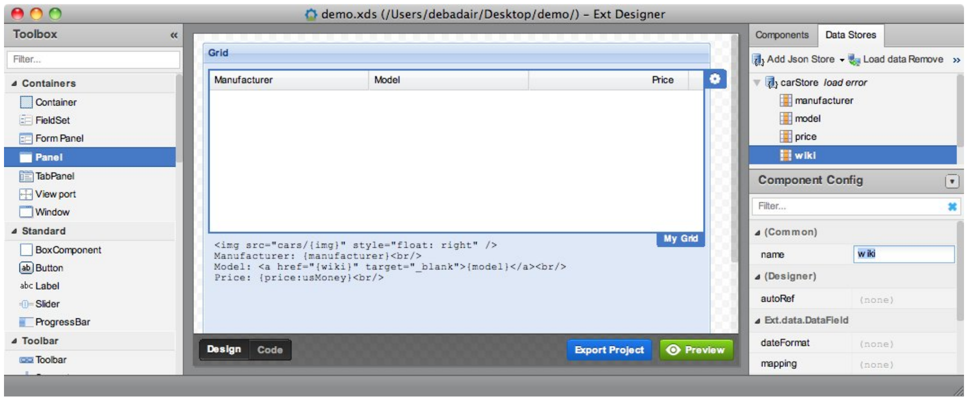
```
<img src="cars/{img}" style="float: right" />
Manufacturer: {manufacturer}<br/>
Model: <a href="{wiki}" target="_blank">{model}</a><br/>
Price: {price:usMoney}<br/>
```

root	data
url	cars/cars.json
jsClass	carStore
userXType	(none)

   b. Set the root attribute to *data*.

   c. Configure the data store to load automatically by enabling the autoLoad attribute. (Otherwise, you won't see any data when you view the index.html file.)

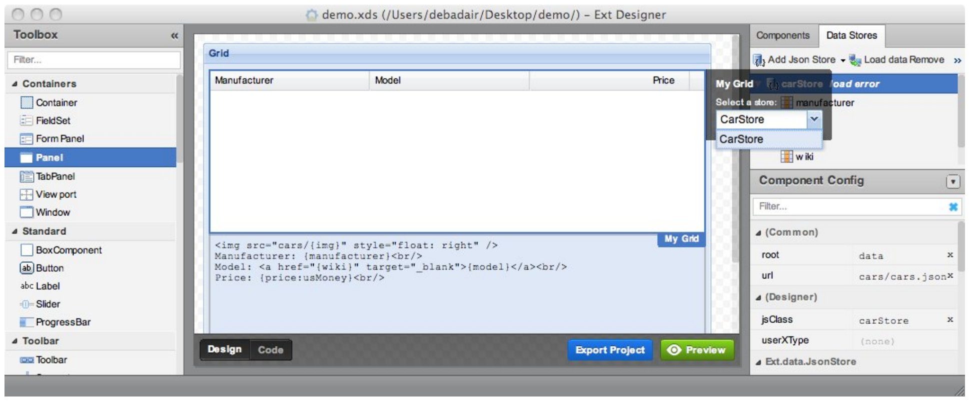
   d. Set the name attribute of each data field in the store: manufacturer, model, price, wiki, and img.



```
<img src="cars/{img}" style="float: right" />
Manufacturer: {manufacturer}<br/>
Model: <a href="{wiki}" target="_blank">{model}</a><br/>
Price: {price:usMoney}<br/>
```

(Common)	
name	wiki
(Designer)	
autoRef	(none)
Ext.data.DataField	
dateFormat	(none)
mapping	(none)

4. Click the flyout config button on the grid component and select CarStore to bind the grid component to the store.



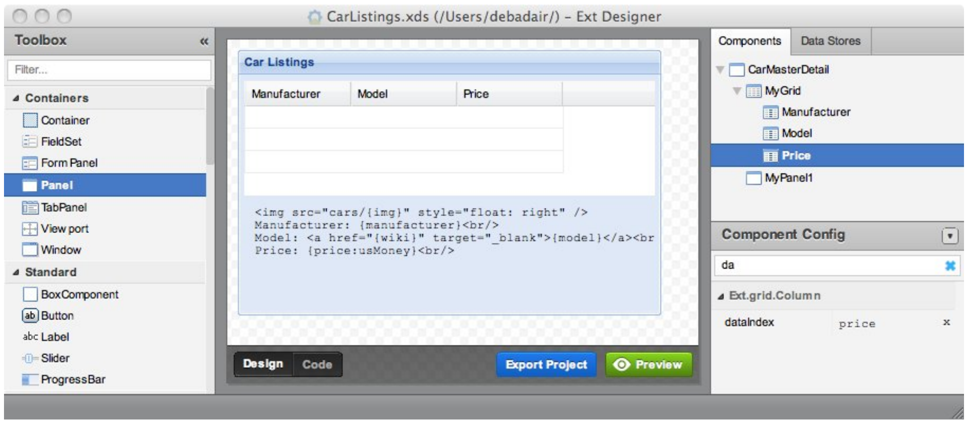
```
<img src="cars/{img}" style="float: right" />
Manufacturer: {manufacturer}<br/>
Model: <a href="{wiki}" target="_blank">{model}</a><br/>
Price: {price:usMoney}<br/>
```

(Common)	
root	data
url	cars/cars.json
(Designer)	
jsClass	carStore
userXType	(none)
Ext.data.JsonStore	

5. Link the columns to the appropriate data fields:

   a. Select a column in the Components list.

   b. Set the dataIndex attribute to the name of the data field.



```
<img src="cars/{img}" style="float: right" />
Manufacturer: {manufacturer}<br/>
Model: <a href="{wiki}" target="_blank">{model}</a><br
Price: {price:usMoney}<br/>
```

Ext.grid.Column	
dataIndex	price

Data from the store is immediately displayed in the grid.

Exporting a Project

Exporting a project generates the Javascript files for your application. When you export, two Javascript files are created for each top-level component in your UI:

- .ui.js contains the base class that defines the component. For example, CarMasterDetailUi. You *extend* this base class to implement your event handler code and custom methods, you do not modify this file directly. The .ui.js files are overwritten whenever you re-export your project.
- .js is a starter file for your implementation. The .js files are generated the first time you export you project. You edit this file to add your event handler code and custom methods.

Important! DO NOT modify the .ui.js file generated by Designer, it will be overwritten whenever you modify and export your project.

Along with the Javascript files, Designer generates an xds_index.html file that loads the javascript and displays your app.

To export your project:

1. Save your changes. (You have to save your project before you'll be able to export it.)
2. Click the Export button below the canvas. The project will be saved to the Export Path specific in the Project Settings. (To change the location, select Edit Preferences from the Edit menu.)

Attaching Event Handlers to UI Components

You can import the files Designer generates into the editor or IDE of your choice. To add event handlers, you need to edit the .js files.

For example, to add an event handler to the Car Listings app that displays the appropriate image and wiki information when a row is selected in the grid:

1. Edit CarMasterDetail.js.
2. Create a selection model for the grid:

   ```
   var sm = this.grid.getSelectionModel();
   ```

   The default selection model for a grid is a RowSelectionModel. Whenever a row in the grid is selected, a rowselect event is fired. This event includes the SelectionModel, rowIndex and the Record that provides the data for the selected row.
3. Add an event handler to call a custom onRowSelect function when a row in the grid is selected:

   ```
   sm.on('rowselect', this.onGridRowSelect, this);
   ```
4. Implement onRowSelect to update the the row with the data from the data store:

   ```
   onGridRowSelect: function(sm, rowIdx, r) {
       this.detail.update(r.data);
   }
   ```

For more information about working with Ext JS grids, see the API documentation.