

Dojo 1.0 Tutorial

Bibhas Bhattacharya, Web Age Solutions Inc.

www.webagesolutions.com

This tutorial shows how to use Dojo 1.0 to create a rich Internet application. To do this tutorial, you need one of the following setups:

1. Eclipse WTP with Tomcat. Make sure that you can create a dynamic web project and run it from Tomcat. Or,
2. RAD6 or RAD7.

Getting Started

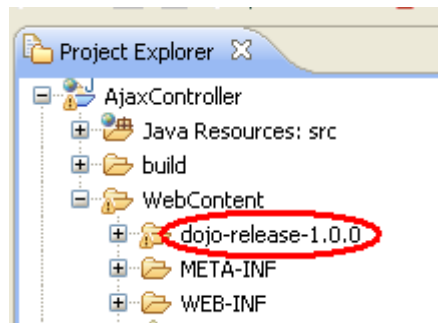
Launch WTP or RAD. Create a dynamic web project. In this tutorial, we call the project AjaxController. You may want to do the same.

Deploy the project to the server.

Install Dojo

Download Dojo 1.0 from: <http://download.dojotoolkit.org/release-1.0.0/dojo-release-1.0.0.tar.gz>.

Extract it in the **WebContent** folder of the dynamic web project.



Back in WTP or RAD, refresh the dynamic web project. You should see the **dojo-release-1.0.0** folder under WebContent.

Tutorial 1 - First Application

We will develop a simple Dojo application. Later, we will analyze the code.

Create the Page

In the WebContent folder create a new HTML page called app1.html.

Set the contents of the page to as follows.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<title>App1</title>

<link rel="stylesheet"
      type="text/css"
      href="dojo-release-1.0.0/dijit/themes/tundra/tundra.css"/>

<script type="text/javascript"
        djConfig="parseOnLoad: true"
        src="dojo-release-1.0.0/dojo/dojo.js">
</script>

<script type="text/javascript">
    dojo.require("dijit.form.DateTextBox");
</script>
</head>

<body class="tundra">
<h1>Application One</h1>
<p>Enter a date below:</p>
<input dojoType="dijit.form.DateTextBox"/>

</body>

</html>
```

Save changes.

Test

Publish the files to the server.

Run the page on the server. For WTP, the URL will be <http://localhost:8080/AjaxController/app1.html>.

Application One

Enter a date below:

Click on the date input box. This will pop open a calendar widget.

Enter a date below:

November						
S	M	T	W	T	F	S
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	1
2	3	4	5	6	7	8
2006		2007	2008			

Congratulations. You have finished the first Dojo application.

Analyze the Code

We will now look at each key aspect of the HTML file.

```
<link rel="stylesheet"
      type="text/css"
      href="dojo-release-1.0.0/dijit/themes/tundra/tundra.css"/>
```

This imports the `tundra.css` stylesheet. Dojo 1.0 has the notion of themes. In this application, we are using the `tundra` theme.

```
<script type="text/javascript"
        djConfig="parseOnLoad: true"
        src="dojo-release-1.0.0/dojo/dojo.js">
</script>
```

This imports the core Dojo library from `dojo.js`. The `djConfig` attribute allows us to set various configuration properties of Dojo. Here, we set the `parseOnLoad` property to `true`. This causes Dojo to attach an `onload` event handler for the body. From that event handler, Dojo traverses the DOM tree and instantiates the widgets whenever a widget markup is located. We will soon find out how to create a widget. **Note:** The default value of `parseOnLoad` is `false`. If you do not set it to `true`, by default, system will not create any widgets.

```
<script type="text/javascript">
    dojo.require("dijit.form.DateTextBox");
</script>
```

The `dojo.require()` function imports the JavaScript code for a specific package. This allows you to import only the portion of Dojo that you need. In this case, we are importing the code for the `dijit.form.DateTextBox` widget. **Note:** To use any widget in a page, you must import its code using `dojo.require()`.

```
<body class="tundra">
```

In this line, we are setting the theme for the application. As you can see, the process is quite simple. We set the class of the body to the name of the theme – "tundra" in this case.

```
<input dojoType="dijit.form.DateTextBox"/>
```

This markup inserts a DateTextBox widget in the page. All form widgets are added using the <input> or <select> tag and the dojoType attribute. When Dojo traverses the DOM tree, it detects the existence of the dojoType attribute. This causes Dojo to create the widget by creating additional DOM elements near the <input> element.

That's pretty much all it takes to create a page with Dojo widgets.

Add a Combo Box Widgets

A combo box is a drop down menu item, where user can edit the value. HTML does not provide any input markup for this. Dojo saves the day by giving us this widget.

Below the line:

```
dojo.require("dijit.form.DateTextBox");
```

Add the following line to import the code for combo box:

```
dojo.require("dijit.form.ComboBox");
```

Below the line:

```
<input dojoType="dijit.form.DateTextBox" />
```

Add:

```
<br/>
<select dojoType="dijit.form.ComboBox"
    autocomplete="true" value="Enter a state">
    <option selected="selected">California</option>
    <option>Illinois</option>
    <option>New York</option>
    <option>Nebraska</option>
</select>
```

As you can see, the combo box is created using the same approach as a drop down menu - a <select> HTML tag that contains a bunch of <option> tags. The only difference is the existence of the dojoType attribute.

Save changes.

Publish files.

Test

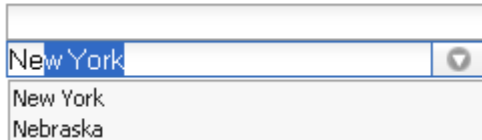
Refresh the web browser page.

Application One

Enter a date below:



You should be able to pick a state from the drop down or enter any other name.



Autocompletion is enabled. If you enter "Ne", system will list the states that begin with that string pattern.

Add Checkbox and Radio Button

HTML provides native support for these widgets. The only reason to use Dojo's equivalent widgets is to get a consistent look and feel.

Add these dojo.require statements below the existing ones.

```
dojo.require("dijit.form.CheckBox");
```

The CheckBox widget also covers radio button. There is no need to import additional code for radio button.

Below the line:

```
</select>
```

Add these lines:

```
<br/>
<input id="ch1" dojoType="dijit.form.CheckBox"
      checked="checked" value="Y"/>
<label for="ch1">Send me e-mail</label>

<br/>
<input id="rad1" dojoType="dijit.form.RadioButton"
      checked="checked" name="vendor" value="IBM"/>
<label for="rad1">IBM</label>
<input id="rad2" dojoType="dijit.form.RadioButton" name="vendor" value="MS"/>
<label for="rad2">Microsoft</label>
<input id="rad3" dojoType="dijit.form.RadioButton" name="vendor" value="Oracle"/>
<label for="rad3">Oracle</label>
```

The check box should be pretty straightforward. The radio button is a bit tricky. The dojoType is dijit.form.RadioButton. In addition, the name attribute must be used. All radio buttons in the group must have the same name ("vendor" in this case).

Save changes.

Publish.

Test

Refresh the browser.

Application One

Enter a date below:

 Send me e-mail
 IBM Microsoft Oracle

Make sure that the checkbox and radio buttons work properly.

Tutorial 2 - Event Handling

Add Buttons

Add a new `dojo.require()` statement to include the code for the `dijit.form.Button` widget.

```
<script type="text/javascript">  
    dojo.require("dijit.form.DateTextBox");  
    dojo.require("dijit.form.ComboBox");  
    dojo.require("dijit.form.CheckBox");  
    dojo.require("dijit.form.Button");  
</script>
```

Just above the line:

```
</body>
```

Add:

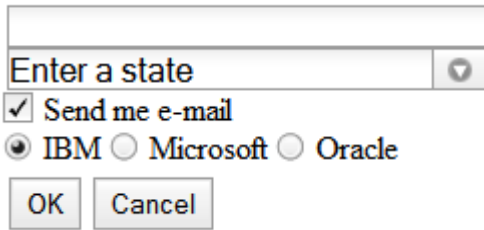
```
<br />  
<button dojoType="dijit.form.Button">OK</button>  
<button dojoType="dijit.form.Button">Cancel</button>
```

Save changes.

Refresh the web browser. Make sure the button shows up.

Application One

Enter a date below:



Enter a state

Send me e-mail

IBM Microsoft Oracle

OK Cancel

Understanding Dojo's Event Handling

Event handling works somewhat differently in Dojo than regular DOM. First, let's recap DOM's event handling scheme.

For a regular button element, you will register an onclick event handler as follows:

```
<button onclick="showEventStd(this, event);">OK</button>
```

The "this" and "event" keywords tell the browser to supply the element and the event object as parameters to the handler function. The handler function looks something like this:

```
function showEventStd(theElement, theEvent) {  
    alert("Event " + theEvent.type + " for element " + theElement.nodeName);  
}
```

In Dojo, there are two main differences.

1. A Dojo button's onclick attribute takes the name of the event handler method. You don't actually call the method like you do for a regular DOM button.
2. Second, the event handler method gets only one parameter – the Event object.

Let's have a look at an example. The event handler is registered as follows:

```
<button dojoType="dijit.form.Button" onclick="showEvent">OK</button>
```

The event handler method looks like this:

```
function showEvent(theEvent) {  
    alert("Event " + theEvent.type);  
}
```

Make sure that you understand the two key differences.

The Event object passed to the handler function is a customized version of the DOM Event object. Some browsers do not strictly follow the W3C DOM specification. For example, the DOM Event object should have a property called target which points to the Element object where the event took place. Mozilla follows this properly. IE, however, calls this property srcElement. Dealing with such differences can be tricky. Dojo solves the problem by supplying the missing standard properties. For example, the Dojo Event object will have the target property in all browsers. So, the following event handler function will work for all browsers.

```
function showEvent(theEvent) {
    alert("Event " + theEvent.type + " for element " + theEvent.target.nodeName);
}
```

Add Button Event Handler

Register event handlers for the buttons as shown below.

```
<button dojoType="dijit.form.Button" onclick="showEvent">OK</button>
<button dojoType="dijit.form.Button" onclick="showEvent">Cancel</button>
```

Write the handler method as shown in bold face below:

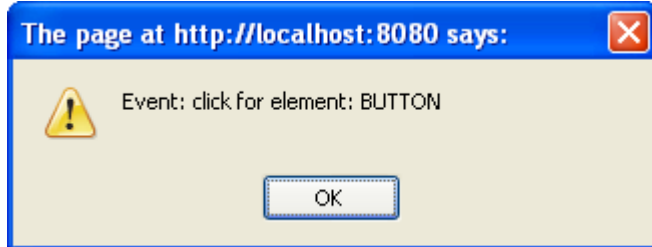
```
<script type="text/javascript">
    dojo.require("dijit.form.DateTextBox");
    dojo.require("dijit.form.ComboBox");
    dojo.require("dijit.form.CheckBox");
    dojo.require("dijit.form.Button");

    function showEvent(theEvent) {
        alert("Event: " + theEvent.type +
            " for element: " + theEvent.target.nodeName);
    }
</script>
```

Save changes.

Refresh the browser.

Click the OK button. The alert dialog will look like this:



Dojo Debugging

Using alert() for debugging can get tiresome. Dojo has a built in logging mechanism. It's pretty easy to use.

First, enable debugging. This is done by setting the isDebug property of Dojo configuration to true.

```
<script type="text/javascript" djConfig="parseOnLoad: true, isDebug: true"
    src="dojo-release-1.0.0/dojo/dojo.js">
</script>
```

Next, replace the alert() function with console.debug().

```
function showEvent(theEvent) {
    console.debug("Event: " + theEvent.type +
        " for element: " + theEvent.target.nodeName);
}
```


Save changes.

Refresh the browser.

The debug console becomes visible at the bottom of the page.

The screenshot shows a web form with the following elements: a text input field, a dropdown menu labeled "Enter a state", a checked checkbox labeled "Send me e-mail", and three radio buttons labeled "IBM", "Microsoft", and "Oracle". Below the form are "OK" and "Cancel" buttons. The "OK" button is highlighted with a dotted border. At the bottom of the form, there is a "Clear" button and a "Close" button.

Click the buttons and make sure that the debug messages show up.

The screenshot shows the same web form as above, but with a debug console visible at the bottom. The debug console shows three messages: "Event: click for element: BUTTON", "Event: click for element: BUTTON", and "Event: click for element: BUTTON".

Handle More Events

The check box and radio buttons also fire the onclick event. Register the showEvent function as event handler as shown below.

```
<input id="ch1" dojoType="dijit.form.CheckBox" checked="checked"
  value="Y" onclick="showEvent"/>
<label for="ch1">Send me e-mail</label>

<br />
<input id="rad1" dojoType="dijit.form.RadioButton" checked="checked"
  name="vendor" value="IBM" onclick="showEvent"/>
<label for="rad1">IBM</label>
<input id="rad2" dojoType="dijit.form.RadioButton" name="vendor"
  value="MS" onclick="showEvent"/>
<label for="rad2">Microsoft</label>
<input id="rad3" dojoType="dijit.form.RadioButton" name="vendor"
  value="Oracle" onclick="showEvent"/>
<label for="rad3">Oracle</label>
```

The combo box and date box, on the other hand, fire onchange event. The onchange event handler

works differently. It does not receive the Event object. Instead, it receives the new value entered by the user.

Register the showNewValue function as the onchange event handler for the combo box and date box as shown in bold face below.

```
<input dojoType="dijit.form.DateTextBox" onchange="showNewValue"/>

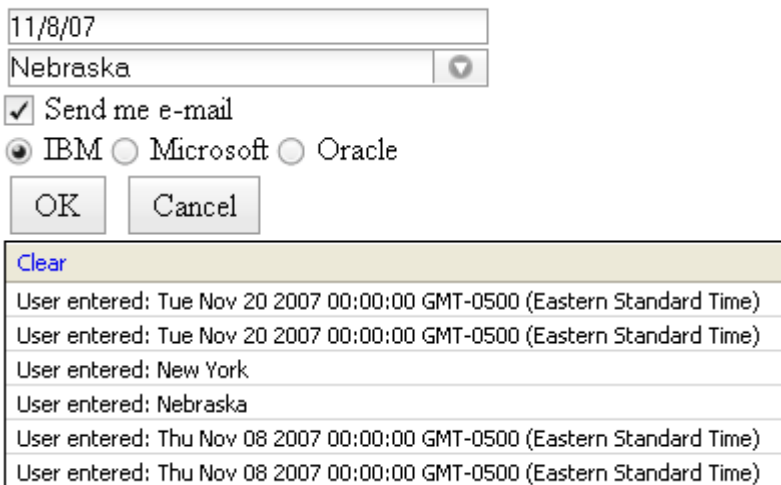
<br />
<select dojoType="dijit.form.ComboBox" autocomplete="true"
  value="Enter a state" onchange="showNewValue">
  <option selected="selected">California</option>
  <option>Illinois</option>
  <option>New York</option>
  <option>Nebraska</option>
</select>
```

Develop the showNewValue function below the showEvent function.

```
function showNewValue(val) {
    console.debug("User entered: " + val);
}
```

Save changes.

Refresh the browser. Make sure that the date box and combo box events are firing properly.



Tutorial 3 – Screen Layout

A rich Internet application will have desktop like screen layout. To that end, Dojo provides layout manager widgets.

Create a Template Page

To help speed up our tutorial, we will create a boiler plate HTML file.

Create a new HTML file called template.html.

Set its contents to as follows.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>

<title>Template</title>

<link rel="stylesheet" type="text/css"
      href="dojo-release-1.0.0/dijit/themes/tundra/tundra.css" />

<script type="text/javascript"
        djConfig="parseOnLoad: true, isDebug: true"
        src="dojo-release-1.0.0/dojo/dojo.js">
</script>

<script type="text/javascript">
    dojo.require("dijit.form.Button");
</script>

</head>

<body class="tundra">

</body>

</html>
```

Save and close the file.

The ContentPane Widget

A content pane is the simplest layout manager. It is just a grouping of other widgets. It does not enforce any particular layout. In a way, it acts just like a `<div>` element. ContentPane does have a powerful feature. It can pull in content from an external URL and populate its internal content asynchronously. This allows you to load content into the pane on demand.

Copy `template.html` and paste it as `content_pane.html`.

Open `content_pane.html`.

Add a `dojo.require()` statement load in the code for the widget. This is shown in boldface below.

```
<script type="text/javascript">
    dojo.require("dijit.form.Button");
    dojo.require("dijit.layout.ContentPane");
</script>
```

Within the body tag, add a content pane widget as shown below in bold face.

```
<body class="tundra">
```

```
<div dojoType="dijit.layout.ContentPane"
      style="background: red">
<p>Hello World</p>
</div>
```

</body>

Save changes.

Run the HTML file on the server to view it in a browser. It should look like this.



Clearly, nothing too exciting. The widget is behaving just like a regular <div> element.

Now, we will externalize the content.

Create a new HTML file called content1.html. Set the contents of the file to:

```
<h3>This is content 1</h3>
<p>Some content here</p>
```

Save changes.

Back in the content_pane.html, change the widget markup as shown below.

```
<div dojoType="dijit.layout.ContentPane"
      style="background: red"
      href="content1.html">
```

Loading ...

</div>

Save changes.

Refresh the browser.

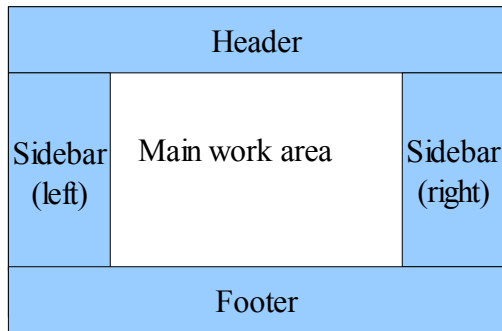


When the main page (content_pane.html) first loads, the ContentPane widget shows "Loading...". This gets eventually replaced by the contents of content1.html.

The LayoutContainer Widget

Most desktop applications have this layout:

1. The top layer has menu and toolbar.
2. There are sidebars on left and/or right side.
3. There is a status bar and widgets at the bottom.
4. In the middle, we have the main work area.



The same pattern happens in most web sites, where we have header, footer, sidebar and main content area.

The LayoutContainer captures this pattern. Each box is represented by a layout widget, such as a ContentPane. Each box has a position, which can be one of top, bottom, left, right and client. The "client" position refers to the main work area. It is also the one that grows with the size of the browser window. All other boxes have a fixed width (for left and right) or height (for top and bottom).

Nothing beats a hands on example. So let's get started.

Copy template.html and paste it as layout_container.html.

Add two dojo.require() statements as shown below in boldface.

```
<script type="text/javascript">
    dojo.require("dijit.form.Button");
    dojo.require("dijit.layout.ContentPane");
    dojo.require("dijit.layout.LayoutContainer");
</script>
```

Within the <body> element, add the layout container as follows.

```
<div dojoType="dijit.layout.LayoutContainer">

    <div dojoType="dijit.layout.ContentPane" layoutAlign="top"
        style="background:red;height:50px">
        Header
    </div>

    <div dojoType="dijit.layout.ContentPane" layoutAlign="bottom"
        style="background:blue">
        Footer
```

```

</div>

<div dojoType="dijit.layout.ContentPane" layoutAlign="left"
      style="background:green;width: 120px;">
Sidebar
</div>

<div dojoType="dijit.layout.ContentPane" layoutAlign="client">
Client area
</div>

</div>

```

The code should be fairly easy to understand.

Now, I must tell you about a strange limitation. In Dojo 1.0, the LayoutContainer widget, the <body> and <html> elements must all have a defined width and height. Otherwise, the page breaks badly (especially in IE). We will solve that problem now.

Within the <head> element, define a style as follows.

```

<style>
.screen {height: 400px; width: 600px; margin: 0; padding: 0;}
</style>

```

Set the class of the <html> element:

```

<html xmlns="http://www.w3.org/1999/xhtml" class="screen">

```

Set the class of body element:

```

<body class="screen tundra">

```

Set the class of the LayoutContainer widget:

```

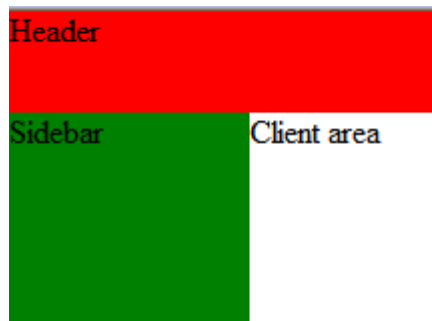
<div dojoType="dijit.layout.LayoutContainer" class="screen">

```

Save changes.

Note: Here, we set a fixed size for the screen. You can also set it to a % figure (such as 75%). This will make the client area grow or shrink as the browser window is resized.

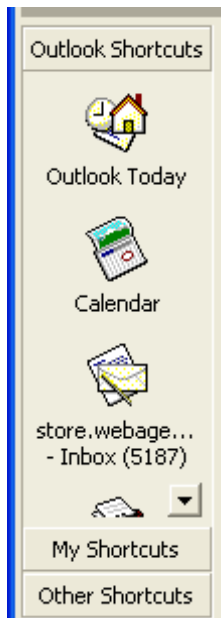
Run the file in the server (http://localhost:8080/AjaxController/layout_container.html).



The AccordionContainer Widget

AccordionContainer can group other widgets in bunches. Only one bunch is visible at a time. This

widget is popularized by MS Outlook. The widget usually appears on the left hand side of the client work area.



Each bunch of widgets is created using an AccordionPane widget.

Add a new `dojo.require()` statement below the existing ones.

```
dojo.require("dijit.layout.AccordionContainer");
```

This will also include code for the AccordionPane widget.

Within the left hand side bar, add the accordion widget as shown below in bold face.

```
<div dojoType="dijit.layout.ContentPane" layoutAlign="left" style="background:green;width: 120px;">
```

```
    <div dojoType="dijit.layout.AccordionContainer">
```

```
        <div dojoType="dijit.layout.AccordionPane" selected="true" title="Tools">
```

```
            <button dojoType="dijit.form.Button">Save</button><br/>
```

```
            <button dojoType="dijit.form.Button">Reload</button>
```

```
        </div>
```

```
        <div dojoType="dijit.layout.AccordionPane" title="Options">
```

```
            <button dojoType="dijit.form.Button">Configure</button><br/>
```

```
            <button dojoType="dijit.form.Button">Session</button>
```

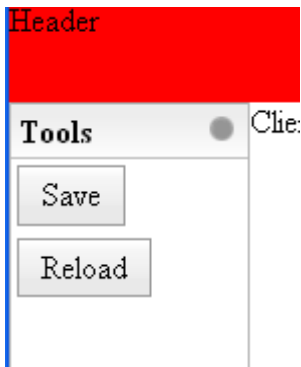
```
        </div>
```

```
    </div>
```

```
</div>
```

Save changes.

Refresh the browser.



You can click Options to view that bunch of widgets.

Because, not all AccordionPane widgets show their contents all the time, you can load their contents on demand. Just set the href attribute to an URL. When the AccordionPane is first expanded, system will fetch the content on demand. Example:

```
<div dojoType="dijit.layout.AccordionPane" title="Options" href="options.html">
</div>
```

The TabContainer Widget

Showing tabbed content is very easy. Each tab is created using a ContentPane. The title of the tab is set using the title attribute of the ContentPane.

Copy template.html as tab.html.

Open tab.html.

Add two dojo.require() elements to include TabContainer and ContentPane.

```
<script type="text/javascript">
    dojo.require("dijit.form.Button");
    dojo.require("dijit.layout.ContentPane");
    dojo.require("dijit.layout.TabContainer");
</script>
```

Within the body tag, add the TabContainer as follows.

```
<div dojoType="dijit.layout.TabContainer"
    style="width:500px;height:100px">

    <div dojoType="dijit.layout.ContentPane" title="Tab A">
        The content of tab A.
    </div>

    <div dojoType="dijit.layout.ContentPane" title="Tab B" href="content1.html">
        Loading...
    </div>

    <div dojoType="dijit.layout.ContentPane" title="Tab C"
        closable="true" selected="true">
        The content of tab C.
    </div>
```

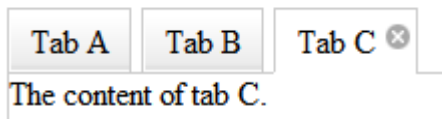

</div>

Note:

1. The content of Tab B is fetched on demand from an external file.
2. Tab C can be deleted (closable is set to true).
3. Tab C is opened by default (selected is set to true).

Save changes.

Run the file in server (<http://localhost:8080/AjaxController/tab.html>).



Tutorial 4 – Using the Grid Widget

TurboAjax Inc. has donated their commercial TurboGrid XHTML widget to Dojo. It has been integrated within Dojo as the `dojox.Grid` widget. The existence of "dojox" in the package name indicates its still in experimental stage. However, the widget, in its original form (TurboWidget) is a sophisticated one. We will start small and slowly expose the more advanced features.

In this tutorial, we will create the simplest possible grid, that looks like this.

Product Title	Price	Type
Baseball gloves	12.34	Sports
Tennis ball	5.99	Sports
T-shirt	12.45	Clothing
Hat	12.45	Clothing

Get Started

Copy `template.html` and paste it as `grid.html`.

Open `grid.html`.

Add a new `dojo.require()` statement as shown in bold face below.

```
dojo.require("dijit.form.Button");  
dojo.require("dojox.grid.Grid");
```

Note: Although, the name of the widget is `dojox.Grid`, you need to import `dojox.grid.Grid`. This may be just an inconsistency that will be corrected in future.

Import a special CSS file, just for the grid widget. **Note:** This CSS file is absolutely necessary. Without

it, the grid widget will simply fail to render properly.

```
<link rel="stylesheet" type="text/css"
      href="dojo-release-1.0.0/dijit/themes/tundra/tundra.css" />
<link rel="stylesheet" type="text/css"
      href="dojo-release-1.0.0/dojox/grid/_grid/tundraGrid.css" />
```

Create the Display Structure

The grid widget can help you create very complex tables. Unfortunately, that means, you need to take a bit of time to learn how it works, even when you are creating a very basic table.

A **view** is an independently scrollable vertical unit. In most cases, we need to scroll the whole table as a unit. Consequently, in most cases, there will be a single view per table.

A view is a collection of **rows**. That's the easy part. We all know that a table is a collection of rows and that a row has several columns. However, in a grid, a row may have many **subrows** and each subrow can have a different number of columns.

Product Title (Subrow 1, Column 1)	Price (Subrow 1, Column 2)	Type (Subrow 1, Column 3)
Description (Subrow 2, Column 1, Colspan 3)		

Here, we see a single row that has two subrows. The top subrow has three columns. The bottom one has one. This single column spans three columns. This row structure will be repeated for every row in the view.

In our case, we have a very simple table. There will be only one subrow per row. That subrow will have three columns – Product Title, Price and Type.

Product Title (Subrow 1, Column 1)	Price (Subrow 1, Column 2)	Type (Subrow 1, Column 3)
---	-----------------------------------	----------------------------------

Now, we will define this table structure.

Below the line:

```
dojo.require("dojox.grid.Grid");
```

Add:

```
var subrow1 = [{name: "Product Title"}, {name: "Price"}, {name: "Type"}];
```

Note, how we are adding three columns to the subrow. Each column object has a **name** property that defines the header title of the column.

Then, add the following line to define the view.

```
var view = {rows: [ subrow1 ]};
```

In our case, the view has only one subrow. The view object has a property called rows that takes an array of subrows. **Note:** The name of the property is confusing. Ideally, it should have been called subrows. If it helps at all, you can also use the **cells** property in place of rows to the same effect.

Finally, define the whole grid structure. In our case, we have only one view.

```
var structure = [ view ];
```

In the <body> tag, add the grid as follows.

```
<body class="tundra">  
  
<div dojoType="dojox.Grid" autoWidth="true" structure="structure"></div>  
  
</body>
```

The autoWidth attribute is important. This automatically sets the width of the grid to a reasonable value. Without it, the grid takes up the whole width of the page and looks really bad.

Save changes.

Run the file in the server.



Product Title	Price	Type
---------------	-------	------

Make sure that the structure of the grid looks good.

Supply Data for the Grid

Data is supplied as a simple JavaScript array. It is an array of rows. Each row, contains an array of cell data.

Add the following script code:

```
var data = [  
    ["Baseball gloves", 12.34, "Sports"],  
    ["Tennis ball", 5.99, "Sports"],  
    ["T-shirt", 12.45, "Clothing"],  
    ["Hat", 12.45, "Clothing"]  
];  
  
var productModel = new dojox.grid.data.Table(null, data);
```

Register the model with the grid:

```
<div dojoType="dojox.Grid" autoWidth="true" model="productModel"  
    structure="structure"></div>
```

Save changes.

Run the file in the server.

Product Title	Price	Type
Baseball gloves	12.34	Sports
Tennis ball	5.99	Sports
T-shirt	12.45	Clothing
Hat	12.45	Clothing