

EWD

Using the Sencha Touch Custom Tags for Mobile Applications

Tutorial: Part 2

Build 846

Background

This is the second part of our tutorial on mobile web application development using Enterprise Web Developer (EWD)'s Sencha Touch Custom Tags.

It is recommended that you complete (or at least read) Part 1 before embarking on the exercises in this second part of the tutorial.

In Part 1, we covered the basics of creating mobile web applications using EWD and Sencha Touch, and touched on how EWD integrates seamlessly with the Cache and GT.M databases.

In this part we'll start to flesh out our example to use some of the cool widgets that EWD and Sencha Touch provides, and we'll begin to create a fully-fledged database-driven application. We'll also examine in more detail the underlying architecture of EWD and in particular what's behind EWD's persistent Sessions.

*Note that for Part 2 you'll need **build 846 or later** of EWD, as some of the features and techniques described in this tutorial have been added since the build originally released for Part 1.*

Lesson 7: Menus and Lists

One of the primary UI components that is used in most mobile web applications is the menu or *List*. The List is a particularly effective way of navigating and selecting from quite large amounts of data because you can use a swipe gesture to quickly scroll through the list. The scrolling list will have momentum and will continue to scroll by after fast swipe gesture.

So let's change our simple Hello World application and make it start with a simple list of 3 options, the first of which will take us to our Hello World page.

What we use to create menus in EWD is the Sencha Touch `<st:list>` tag. The options or data in the list is defined dynamically within Cache or GT.M. So create a new file in your *stdemo* directory named *mainMenu.ewd* as follows:

```
<ewd:config isFirstPage="false" pageType="ajax" prepagescript="getMainMenu^stdemo">
<st:panel fullscreen="true" scroll="vertical">
  <st:toolbar id="topToolbar" dock="top" title="EWD Demo">
  </st:toolbar>

  <st:list id="mainMenu" sessionName="mainMenuOptions" scroll="false">
    <st:layout>
      <st:field name="optionText" displayInList="true"/>
    </st:layout>
  </st:list>

</st:panel>
```

Now add the Pre-Page script function *getMainMenu()* to your *stdemo* routine, eg:

```
getMainMenu(sessid)
;
n list
;
s list(1,"optionText")="Hello World"
s list(2,"optionText")="Example 2"
s list(3,"optionText")="Example 3"
;
;
d saveListToSession^%zewdSTAPI(.list,"mainMenuOptions",sessid)
;
QUIT ""
```

Finally we need to modify the *index.ewd* page so that it loads up the new *mainMenu.ewd* page instead of our original *helloworld.ewd* page:

```
<ewd:config isFirstPage="true" cachePage="false">

<st:container rootPath="/sencha-1.0/" contentPage="mainMenu" title="EWD Demo">
  <script src="/stdemo.js" />
  <st:images>
    <st:image type="tabletStartupScreen" src="/sencha-
1.0/examples/kitchensink/resources/img/tablet_startup.png" />
    <st:image type="phoneStartupScreen" src="/sencha-
1.0/examples/kitchensink/resources/img/phone_startup.png" />
    <st:image type="icon" src="/sencha-1.0/examples/kitchensink/resources/img/icon.png"
addGloss="true" />
  </st:images>
</st:container>
```

Once you've created and saved these files, recompile the *stdemo* application.

If you're using GT.M, you'll also have to ensure that the routine file *stdemo.m* that contains the new pre-page script has been re-compiled, by running, in the Linux shell:

```
mumps stdemo.m
```

If you've done everything correctly, when you re-run the *stdemo* application you should see:



Let's take a look at what we did to create this. Specifically look at *mainMenu.ewd* and its pre-page script (*getMainMenu*):

- First we used the `<st:list>` tag which creates an instance of the *Ext.List* class.
- We added this as a sub-component of the main panel which is occupying the full screen of our mobile device, so it sat inside the main panel and below the top toolbar
- The pre-page script creates a simple array of options. The first subscript (1..n) denotes the position of the option in the list, and the second subscript ("*optionText*") defines a property that we want to use and/or display in the list.
- The `<st:list>` tag has a child tag `<st:layout>` which allows us to define how we want each row in the list to be presented. In this example we just want to display the *optionText* property, so we use the `<st:field>` tag to denote that.
- The option array that is created in the pre-page script is saved to the EWD Session as a JSON datastore named *mainMenuOptions* using the special EWD/Sencha Touch API: *saveListToSession()*
- By defining the *sessionName* attribute of the `<st:list>` tag to be *mainMenuOptions*, the list of options we saved into the EWD Session will be used to dynamically populate the List.

Note that we could have used any property name instead of *optionText*, but whatever name you use in the options array must be referenced as an `<st:field>` name within the `<st:layout>` tag.

Adding a handler

OK so we've now displayed our list of options as a menu. However, it's not doing much yet: tapping any of the options will have no effect at present.

That's because we haven't defined any handler for the menu options yet. So let's change that. As you're going to discover, we have numerous options available to us now, but let's start with something simple and we'll progressively make it slicker and powerful.

First edit the *mainMenu.ewd* page as follows:

```

<ewd:config isFirstPage="false" pageType="ajax" prepagescript="getMainMenu^stdemo">
<st:panel id="mainPanel" fullscreen="true">
  <st:toolbar id="topToolbar" dock="top" title="EWD Demo">
  </st:toolbar>

  <st:list id="mainMenu" sessionName="mainMenuOptions" onTap="EWD.sencha.helloWorld">
    <st:layout>
      <st:field name="optionText" displayInList="true"/>
    </st:layout>
  </st:list>

  <st:panel id="helloworld" html="Hello World!" hidden="true" />
</st:panel>

```

What we've done is add an *onTap* attribute to the `<st:list>` tag and we've added our *Hello World* panel as an initially hidden sub-panel.

Now let's add that *onTap* handler function to the *stdemo.js* file:

```

EWD.sencha.helloWorld = function(index,record) {
  Ext.getCmp("mainMenu").hide();
  Ext.getCmp("helloworld").show();
};

```

The *onTap* handler function takes two parameters: *index* (the option number 0..n), and *record* (the data record held in the tapped row)

In this first example, we'll hide the main menu and make the *Hello World* panel appear in its place. Remember that the *id* attribute of our Sencha Touch tags allows us to point directly to them using Sencha Touch's *Ext.getCmp()* method.

If you save the edited *stdemo.js* file, recompile the modified *mainMenu.ewd* page and re-run the application, you should now find that tapping any of the menu options will make the menu disappear and the *Hello World!* Message will appear.

OK so that's a step forwards, but how can we make the *Hello World* message appear only if we click the first menu option? That's where the *index* parameter can be useful. Try editing the *EWD.sencha.helloWorld()* function as follows:

```

EWD.sencha.helloWorld = function(index,record) {
  Ext.getCmp("mainMenu").hide();
  If (index === 0) Ext.getCmp("helloworld").show();
};

```

When you save *stdemo.js* and re-run the application, you should now find *Hello World!* only appears when you click the first menu option.

Adding a Slide Animation

That's getting better, but one of the cool features you'll frequently see in mobile applications is the animations such as sliding panels when you tap menu options, so let's now add that to our menu.

The first thing to understand is how animated sliding panels work conceptually in Sencha Touch. What you require is a special type of containing panel. In Sencha Touch, panels can have a property known as a layout, and there are a variety of layout types available. The one we require for sliding animations is called a card layout, and in EWD you can specify this using the `<st:cardPanel>` tag. We need to make both the *List* and our *Hello World* panel to be members of this Card Panel in order for them to be animated. So change *mainMenu.ewd* as follows:

```
<ewd:config isFirstPage="false" pageType="ajax" prepagescript="getMainMenu^stdemo">
<st:panel id="mainPanel" fullscreen="true">
  <st:toolbar id="topToolbar" dock="top" title="EWD Demo">
  </st:toolbar>

  <st:cardPanel id="mainCardPanel">

    <st:list id="mainMenu" sessionName="mainMenuOptions" onTap="EWD.sencha.helloWorld">
      <st:layout>
        <st:field name="optionText" displayInList="true"/>
      </st:layout>
    </st:list>

    <st:panel id="helloworld" html="Hello World!" />

  </st:cardPanel>

</st:panel>
```

What we've done is to wrap the `<st:list>` tag inside a `<st:cardPanel>` tag, and we've also moved the *Hello World* panel inside the `<st:cardPanel>`. Only one panel at a time is displayed in a Card Panel, and the first one is used as the one to initially be displayed. As a result we can safely remove the `hidden="true"` attribute from the *helloworld* panel.

Now modify the `EWD.sencha.helloWorld()` function:

```
EWD.sencha.helloWorld = function(index,record) {
  If (index === 0) Ext.getCmp("mainCardPanel").setActiveItem(Ext.getCmp("helloworld"),"slide");
};
```

The Sencha Touch `setActiveItem()` function selects which panel is to be made visible (or active) inside the Card Panel. Here we'll make the panel with an *id* of *helloworld* the active one, and we specify a slide transition to provide the desired animation.

Save these files, recompile the application and now when you click the first menu option, the *Hello World* panel should slide into view. Now it's starting to look cool!

Using EWD to make the coding even simpler

Although we haven't had to write much code to make this happen, EWD can make specifying menus even quicker and simpler, with less reliance on explicitly using the underlying Sencha Touch Javascript functions.

The first thing we can do is to programmatically specify the page associated with a particular option within our pre-page script. We do that by adding an additional property to the array that defines the page: we'll use a property named *page* for this, but you can choose any name you like:

```
getMainMenu(sessid)
;
n json,list
;
;
s list(1,"optionText")="Hello World"
s list(1,"page")="helloworld"
s list(2,"optionText")="Example 2"
s list(2,"page")="example2"
s list(3,"optionText")="Example 3"
s list(3,"page")="example3"
;
;
d saveListToSession^%zewdSTAPI(.list,"mainMenuOptions",sessid)
;
QUIT ""
```

Now add this new field to the layout, but make sure it's not actually displayed by setting `displayInList="false"`:

```

<ewd:config isFirstPage="false" pageType="ajax" prepagescript="getMainMenu^stdemo">
<st:panel id="mainPanel" fullscreen="true">
  <st:toolbar id="topToolbar" dock="top" title="EWD Demo">
  </st:toolbar>

  <st:cardPanel id="mainCardPanel">

    <st:list id="mainMenu" sessionName="mainMenuOptions" onTap="EWD.sencha.helloWorld">
      <st:layout>
        <st:field name="page" displayInList="false"/>
        <st:field name="optionText" displayInList="true"/>
      </st:layout>
    </st:list>

    <st:panel id="helloworld" html="Hello World!" />

  </st:cardPanel>
</st:panel>

```

Now we can replace that *onTap* attribute with a special EWD attributed named *nextPageField*:

```

<ewd:config isFirstPage="false" pageType="ajax" prepagescript="getMainMenu^stdemo">
<st:panel id="mainPanel" fullscreen="true">
  <st:toolbar id="topToolbar" dock="top" title="EWD Demo">
  </st:toolbar>

  <st:cardPanel id="mainCardPanel">

    <st:list id="mainMenu" sessionName="mainMenuOptions" nextPageField="page">
      <st:layout>
        <st:field name="page" displayInList="false"/>
        <st:field name="optionText" displayInList="true"/>
      </st:layout>
    </st:list>

    <st:panel id="helloworld" html="Hello World!" /> ← remove

  </st:cardPanel>
</st:panel>

```

Next, as it shows above, we're going to take our *Hello World* panel out of the *mainMenu.ewd* page and put it back again into its own EWD file, named *helloworld.ewd*. All it should contain is:

```
<ewd:config isFirstPage="false" pageType="ajax">
<st:panel id="helloworld" html="Hello World!" />
```

We don't have to specify a transition since *slide* is used by default. Just to make sure that our new mechanism will be used instead of our original Javascript function, delete or comment out the *EWD.sencha.helloworld()* function from *stdemo.js*.

Now save the files and recompile *mainMenu.ewd* and try re-running the application.

You should find this behaves just like our first "long-hand" approach.

There's a key difference in this approach: the *helloworld* panel is being injected into the browser's page using Ajax techniques rather than pre-existing in the page but not yet visible. This has its downsides: a round-trip to the server is required to fetch it. However it also has many benefits, not least the fact that it allows us to break up the UI into small, manageable chunks that can, if required, be developed and maintained independently by different people.

Now try trying some different transitions. Possible options are:

- flip
- pop
- cube
- fade
- slide

For example:

```
<st:list id="mainMenu" sessionName="mainMenuOptions" nextPageField="page"
transition="fade">
```

As a final exercise in this part of the lesson, now try adding panels for the Example 2 and Example 3 options by creating separate EWD fragment files for them, ie *example2.ewd* and *example3.ewd*.

You should now have a fully functioning menu, with each of the three options bringing their respective panel into view.

Adding a Back Button

Of course, by now it will be obvious to you that there's still a deficiency in our menu: it's impossible to go back to the main menu and choose a different option. The only way to see the main menu again is to restart the application!

So let's see how that is done. Once again there is the "long-hand" manual Sencha Touch mechanism and a simple automated shortcut EWD mechanism. Let's use the Sencha Touch mechanism first.

The first step is to add a back button into the toolbar in *mainMenu.ewd*:

```
<ewd:config isFirstPage="false" pageType="ajax" prepagescript="getMainMenu^stdemo">
<st:panel id="mainPanel" fullscreen="true">
  <st:toolbar id="topToolbar" dock="top" title="EWD Demo">
    <st:toolbarButton ui="back" id="backBtn" text="Back" hidden="true"
handler="EWD.sencha.onBackBtnTapped" />
  </st:toolbar>

  <st:cardPanel id="mainCardPanel">
    <st:list id="mainMenu" sessionName="mainMenuOptions" nextPageField="page" transition="slide">
      <st:layout>
        <st:field name="page" displayInList="false"/>
        <st:field name="optionText" displayInList="true"/>
      </st:layout>
    </st:list>
  </st:cardPanel>
</st:panel>
```

The *ui="back"* attribute specifies that we want a properly styled Back-button, and we want it initially hidden. We've also specified a handler, so let's define that next.

What we need is a function that will return the user to the mainMenu panel, reversing the slide transition. So let's add that to the *stdemo.js* file:

```
EWD.sencha.onBackBtnTapped = function() {
  Ext.getCmp("mainCardPanel").setActiveItem(0, {type:"slide", direction: 'right'});
  Ext.getCmp("backBtn").hide();
  Ext.getCmp("helloworld").destroy();
};
```

The *setActiveItem()* function is used again to return the List into view: notice that we use a 0 (zero) as its first parameter to specify the first item in the Card Panel which was the List. The second parameter is an object that invokes a reverse slide transition.

When we return to the List, we want to hide the button again which is the purpose of the second command. But what about that third command that uses the *destroy()* function: what's it there for?

The reason is because we're using Ajax techniques to inject the *helloworld* (or the other) panel, and when we go back to the menu there's a good chance that we'll choose the same option and inject another instance of the *helloworld* panel into the page. Because

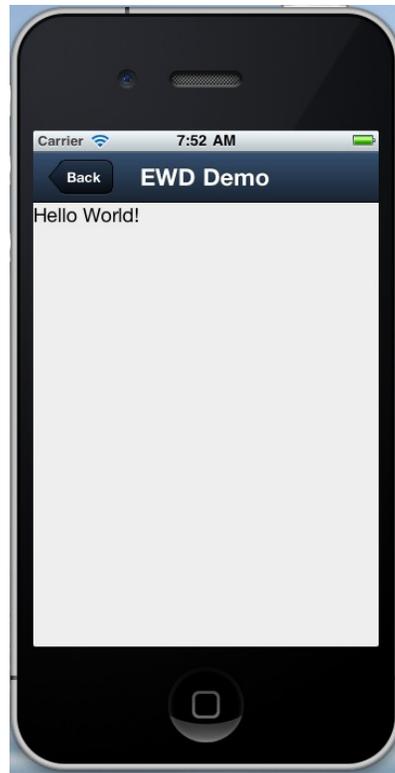
the *helloworld* panel is actually represented as a complex set of Javascript objects within Sencha Touch, it's going to get horribly confused if we try to inject multiple instances of this same object into the page. We can avoid that by getting Sencha Touch to destroy it and clear away all vestiges of it when we go back to the List:

There's one last thing we need to do: make the button appear when the *helloworld* panel slides into view. We're using EWD's automated *nextPageField* mechanism, so we don't have an explicit handler function that we have access to in order to *show()* the back button. However, we can add the necessary Javascript into the *helloworld.ewd* page:

```
<ewd:config isFirstPage="false" pageType="ajax">
<st:js at="end">
  Ext.getCmp("backBtn").show();
</st:js>
<st:panel id="helloworld" html="Hello World!" />
```

This inline Javascript command will make the back button appear when the fragment is injected into the page.

Try it all out: Save the new versions of the files, recompile them and run the application again. You should now be able to navigate to and fro between the *Main Menu* and the *HelloWorld* panels, with the back button appearing and disappearing at the correct times:



We still have a problem: that *destroy()* method is only being applied to the *helloworld* panel. What if the user chooses one of the other options? Things will clearly go wrong. So we need to change that Back Button handler to identify and destroy whatever is the current panel. That's actually quite simple to do: here's the modified version of the handler function:

```
EWD.sencha.onBackBtnTapped = function() {
    var cardPanel = Ext.getCmp("mainCardPanel");
    var currentPanel = cardPanel.getActiveItem();
    cardPanel.setActiveItem(0, {type:"slide", direction: 'right'});
    Ext.getCmp("backBtn").hide();
    currentPanel.destroy();
};
```

Note the way I've set up variables that point to the *cardPanel* and *currentPanel* objects: that's because I'm making more than one reference to them in the logic, so it's more efficient done this way. As a side benefit I think it's also easier to read and maintain when written this way.

The key function here is the Sencha Touch *getActiveItem()* which returns the currently active item in the Card Panel. Now we have a totally general purpose Back Button handler.

Automating the List's Back Button

So that's how to use the Sencha Touch functions to implement the back button with the required behaviour. However, provided we're only using a single level of menus in our application, EWD can automate almost everything all for us. Simply remove the *ui* and *handler* attributes from the back button tag in *mainMenu.ewd* and replace them with `type="autoback"`:

```
<ewd:config isFirstPage="false" pageType="ajax" prepagescript="getMainMenu^stdemo">
<st:panel id="mainPanel" fullscreen="true">
  <st:toolbar id="topToolbar" dock="top" title="EWD Demo">
    <st:toolbarButton type="autoback" id="backBtn" text="Back" hidden="true" />
  </st:toolbar>

  <st:cardPanel id="mainCardPanel">
    <st:list id="mainMenu" sessionName="mainMenuOptions" nextPageField="page" transition="slide">
      <st:layout>
        <st:field name="page" displayInList="false"/>
        <st:field name="optionText" displayInList="true"/>
      </st:layout>
    </st:list>
  </st:cardPanel>
</st:panel>
```

For good measure you can delete or comment out the handler that we wrote in *stdemo.js*. We will still need the inline code that brings the back button into view in the *helloworld* (and other) panels.

Save the files, recompile the application and run it again: it should behave just as before, but this time entirely automatically!

You'll find that this automated mechanism will break down if you have several layers of menus and sub-menus: if so, revert to the explicit handlers we used initially. Also, if you require multiple levels of menus, the simplest approach is to have specific back buttons for each level, each with their own handler, and each brought in and out of view at the appropriate times. You saw how to do that in Part 1.

Lesson 8: Tab Panels

In Lesson 7 we saw how to create a simple menu and Card Panel that we used for navigation. In Sencha Touch there is an alternative UI technique which is the TabPanel. Instead of a list of options in a menu, we can present the options as a set of tabs in a toolbar.

The advantage of this approach is its simplicity: the TabPanel looks after all the animation and to-ing and fro-ing between options for us. If you've only got a small number of menu/navigation options, they're probably the better technique to use.

So let's rewrite our application to use a TabPanel instead.

Create a new EWD page in your **stdemo** directory named *tabPabel.ewd* with the following contents:

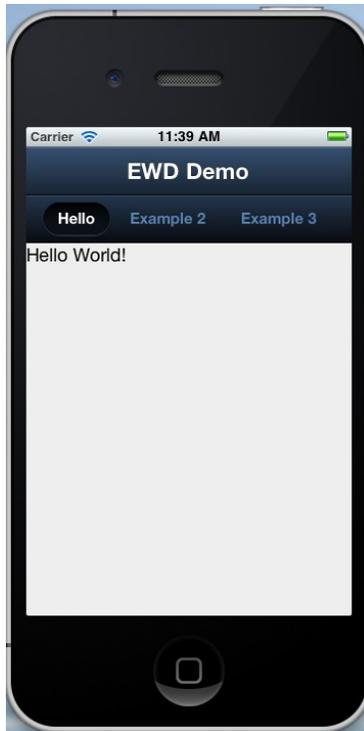
```
<ewd:config isFirstPage="false" pageType="ajax">
<st:panel id="mainPanel" fullscreen="true">
  <st:toolbar id="topToolbar" dock="top" title="EWD Demo" />

  <st:tabPanel id="mainTabPanel">
    <st:panel id="helloworld" title="Hello" html="Hello World!" />
    <st:panel id="example2" title="Example 2" html="Example 2" />
    <st:panel id="example3" title="Example 3" html="Example 3" />
  </st:tabPanel>
</st:panel>
```

Now change *index.ewd* so that it loads this page instead of *mainMenu.ewd*, ie:

```
<ewd:config isFirstPage="true" cachePage="false">
<st:container rootPath="/sencha-1.0/" contentPage="tabPanel" title="EWD Demo">
  <script src="/stdemo.js" />
  <st:images>
    <st:image type="tabletStartupScreen" src="/sencha-1.0/examples/kitchensink/resources/img/tablet_startup.png" />
    <st:image type="phoneStartupScreen" src="/sencha-1.0/examples/kitchensink/resources/img/phone_startup.png" />
    <st:image type="icon" src="/sencha-1.0/examples/kitchensink/resources/img/icon.png"
    addGloss="true" />
  </st:images>
</st:container>
```

Recompile the *stdemo* application and try running it again. This time you should see:



Try clicking the tabs and you'll see the three sub-panels alternately slide in and out of view. This is clearly a much simpler method to specify in order to allow navigation around your applications.

You can see, however, the key limitation: if you're running the application on an iPhone or Android phone: you don't have much room for meaningful tab names. If you are running on a desktop or tablet you have much more room, in which case tabs become a much more useful UI device.

Using EWD Fragments with the Tab Panel

You can also see that in order to use the TabPanel widget, you apparently need to specify all the subpanels and their contents in the same page. This can be advantageous, because no additional round-trips to the server are required.

However, your application may be sufficiently large that you want to break it up into EWD fragments. This is, in fact, possible to do. Let's once again look at some of the manual techniques you could use by using Sencha Touch's own built-in APIs, and then we'll finally look at how EWD can simplify the task to an almost trivial degree.

One Sencha Touch technique we could use is to detect the event that is triggered when a sub-panel is brought into view as a result of you clicking a tab. You could do this in *tabPanel.ewd* as follows:

```

<ewd:config isFirstPage="false" pageType="ajax">
<st:js at="end">

  Ext.getCmp("mainTabPanel").on("cardswitch",function() {
    Ext.Msg.alert('Attention!', this.getActiveItem().id, Ext.emptyFn);
  });

</st:js>

<st:panel id="mainPanel" fullscreen="true">
  <st:toolbar id="topToolbar" dock="top" title="EWD Demo" />

  <st:tabPanel id="mainTabPanel">
    <st:panel id="helloworld" title="Hello" html="Hello World!" />
    <st:panel id="example2" title="Example 2" html="Example 2" />
    <st:panel id="example3" title="Example 3" html="Example 3" />
  </st:tabPanel>

</st:panel>

```

You saw the `<st:js>` tag in Part 1. So this is inline Javascript that will be added after the generated Sencha Touch code (controlled by the `at="end"` attribute), and it is adding a `"cardswitch"` event handler to the Tab Panel. This will fire each time you click one of the tabs and will display the id of the active panel.

If we replaced the `alert` function with the `EWD.ajax.getPage()` function then we should be able to fetch a corresponding fragment for each tab, eg:

```

Ext.getCmp("mainTabPanel").on("cardswitch",function() {
  EWD.ajax.getPage({page: this.getActiveItem().id });
});

```

However, there are a few problems with this technique:

- the `cardswitch` event does not fire when the `tabPanel` page is initially loaded so content for the `helloworld` panel will not be fetched.
- We would only want to fetch the content once for each sub-panel, so we'd have to use some kind of Javascript array to record this, and only call `EWD.ajax.getPage()` if the fragment isn't already logged in the array.

There are other, similar Sencha Touch techniques that you could use, such as detecting the render event for each of the subpanels that you can try experimenting with if you wish.

Some advanced EWD: Using the <st:listener> tag

Rather than manually coding the Javascript within an <st:js> tag, an advanced technique within EWD is to specify a listener for each sub-panel. For example, you could expand the *helloworld* panel as follows:

```
<st:panel id="helloworld" title="Hello">
  <st:listeners>
    <st:listener render=".function() {EWD.ajax.getPage({page:'helloworld'});}" />
  </st:listeners>
</st:panel>
```

Note the period (.) in front of the text string “*function*”. This tells EWD to not use the attribute value as a quoted string literal, but rather as an unquoted reference. If you want to try out this example, you’ll need to make one more change, this time to your *helloworld.ewd* page:

```
<ewd:config isFirstPage="false" pageType="ajax">
<st:panel html="Hello World!" addTo="helloworld" />
```

The reason for this is that this fragment will otherwise simply generate and inject a Javascript constructor to create the *Hello World* panel into the browser page, but it won’t do anything with it. By adding the *addTo* attribute, we’re telling EWD to also add it to the Sencha Touch widget that has an *id="helloworld"* which is the first sub-panel in the Tab Panel. The result is we’ve now created a stub panel in our *TabPanel.ewd* page which automatically fetches its content from the *helloworld.ewd* fragment when the Tab Panel is rendered.

That gives you some insight into some of the advanced techniques that you can use when you become familiar with EWD and Sencha Touch.

Automating it with EWD

You’re probably thinking that things all started to get terribly complicated! Now let’s look at how EWD can simplify everything again for us.

Since this kind of behaviour is something you’ll want to do very frequently, EWD provides an automated way of achieving all this functionality with just a single attribute called *page* on each of the sub-panels. We can therefore re-write *tabPanel.ewd* as follows:

```
<ewd:config isFirstPage="false" pageType="ajax">
<st:panel id="mainPanel" fullscreen="true">
  <st:toolbar id="topToolbar" dock="top" title="EWD Demo" />

  <st:tabPanel id="mainTabPanel">
    <st:panel id="helloworld" title="Hello" page="helloworld" />
    <st:panel id="example2" title="Example 2" page="example2" />
    <st:panel id="example3" title="Example 3" page="example3" />
  </st:tabPanel>
</st:panel>
```

Now edit or create the three content fragment files as follows:

helloworld.ewd:

```
<ewd:config isFirstPage="false" pageType="ajax">
<st:panel html="Hello World!" addTo="helloworld" />
```

example2.ewd:

```
<ewd:config isFirstPage="false" pageType="ajax">
<st:panel html="This is example2" addTo="example2" />
```

example3.ewd:

```
<ewd:config isFirstPage="false" pageType="ajax">
<st:panel html="This is example3" addTo="example3" />
```

Recompile the *stdemo* application and now try running it. If you look in detail at what happens when you run it using Chrome's Developer Tools, you'll see three parallel asynchronous requests are made for the content fragments immediately after *TabPanel.ewd* is rendered.

The cool thing is that these requests are only made once when the Tab Panel is initially rendered, and thereafter when you switch between tabs, you're working entirely locally in the browser.

Lesson 9: Selecting data from a List

We're going to re-visit the `<st:list>` tag, but this time we'll examine it's role in displaying a set of data. We'll also look at how you can do something when the user selects an item from the list.

The differences to our previous example where we used a List as a UI navigation device are:

- The list will contain the results of a database search rather than some predetermined options
- We will probably want to perform the same action irrespective of the row in the List that is selected by the user.

For this example we'll use one of the Globals that EWD will have created on your system, specifically the one that holds the documentation for its DOM APIs. This will give us a nice long list of options to display in our application.

We'll use the Tab Panel that we created in the previous lesson, and we'll replace the middle panel with a panel that will display the list of the APIs. So edit `tabPanel.ewd` as follows:

```
<ewd:config isFirstPage="false" pageType="ajax" prePageScript="getAPIList^stdemo">
<st:panel id="mainPanel" fullscreen="true" layout="card">
  <st:toolbar id="topToolbar" dock="top" title="EWD Demo" />

  <st:tabPanel id="mainTabPanel">
    <st:panel id="helloworld" title="Hello" page="helloworld" />

    <st:panel id="example2" title="List" layout="card">
      <st:list scroll="vertical" id="apiList" sessionName="listOfAPIs">
        <st:layout>
          <st:field name="optionText" displayInList="true"/>
        </st:layout>
      </st:list>
    </st:panel>

    <st:panel id="example3" title="Example 3" page="example3" />
  </st:tabPanel>
</st:panel>
```

When you are creating a List such as this inside a Tab Panel, it's very important to follow a pattern such as the one shown above otherwise Sencha Touch won't display anything. In particular, make sure you have the attributes exactly as shown in red. You'll also notice that we've defined the List within the `tabPanel` page rather than in a separate fragment as per our last lesson. This is because the panel that is to contain the List (`id="example2"`) cannot render a List if the panel is initially empty.

The `<st:list>` tag is defined similarly to our previous example, and this time the list of options is being held in an EWD Session variable named `listOfAPIs`. That is being created by the pre-page script: `getAPIList^stdemo`. Here's what that function looks like:

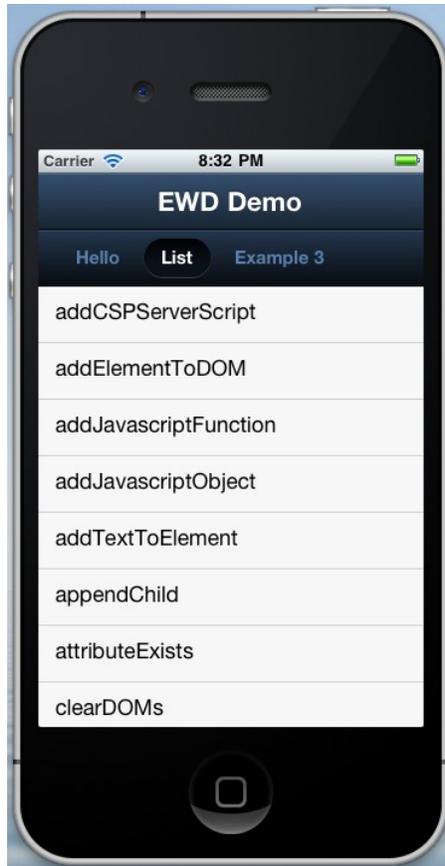
```
getAPIList(sessid)
;
;
n list,name,no
;
s name="",no=0
f s name=$o(^%zewd("documentation","DOM","method",name)) q:name="" d
. s no=no+1
. s list(no,"optionText")=name
;
d saveListToSession^%zewdSTAPI(.list,"listOfAPIs",sessid)
;
;
QUIT ""
```

So the only real difference from our first List example is that the option list is being generated by iterating through a Global.

Save these files, recompile and run the application again and you should see the following:



Clicking on the List tab will bring up the list of APIs:



You can try swiping through the menu to see how quickly you can search and navigate through it.

So that's our menu displaying. Now how can we use it?

Typically you'll want the user to be able to tap a menu option and have a new panel slide into view that presents some more specific information about the option they selected, either to simply view it or manipulate it in some way. Let's see how to do that.

The first thing is to add two EWD-specific attributes to the `<st:list>` tag in the `tabPanel.ewd` page:

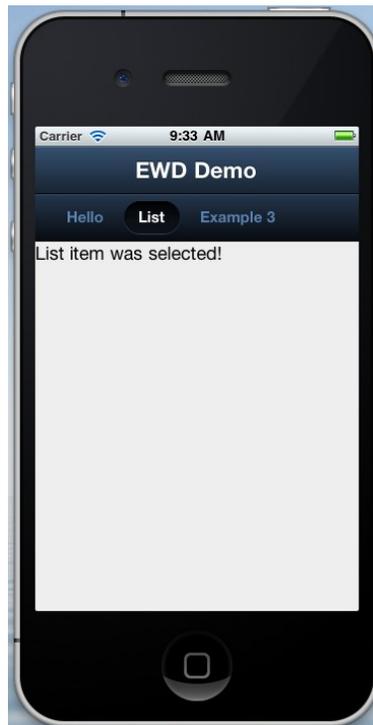
```
<st:panel id="example2" title="List" layout="card">
  <st:list scroll="vertical" id="apiList" sessionName="listOfAPIs" nextPage="apiDetails"
cardpanel="example2">
  <st:layout>
    <st:field name="optionText" displayInList="true"/>
  </st:layout>
</st:list>
</st:panel>
```

The *nextPage* attribute tells EWD the name of the fragment to fetch when any item in the list is tapped. The *cardpanel* attribute tells EWD the *id* of the Card Panel that you want to use so that the *nextPage* fragment can be both added to it and animated as it comes into view. A slide transition is used by default. Notice in *tabPanel.ewd* that we specified the *example2* panel to have a *layout="card"* for this purpose (as highlighted in red above).

We haven't created that fragment named *apiDetails.ewd* yet, so let's do that now: create the file in your **stdemo** directory. Initially we'll keep its contents simple just to check this step works correctly:

```
<ewd:config isFirstPage="false" pageType="ajax">
<st:panel id="apiDetails" html="List item was selected!" />
```

Save, compile and re-run the application as usual. This time when you tap any of the API names in the list, our new *apiDetails* panel should now slide into view:



Of course we now need a Back button in the toolbar to allow us to return to the list of APIs, so let's add that. We use the same mechanism that was described earlier:

Add the toolbar button into *tabPanel.ewd*, make it styled as a Back button, initially hidden and add a reference to its handler function:

```

<ewd:config isFirstPage="false" pageType="ajax" prePageScript="getAPIList^stdemo">

<st:panel id="mainPanel" fullscreen="true" layout="card">
  <st:toolbar id="topToolbar" dock="top" title="EWD Demo">
    <st:toolbarButton ui="back" id="listBackBtn" text="Back" hidden="true"
handler="EWD.sencha.onListBackBtnTapped" />
  </st:toolbar>

  <st:tabPanel id="mainTabPanel">
    <st:panel id="helloworld" title="Hello" page="helloworld" />
    <st:panel id="example2" title="List" layout="card">
      <st:list scroll="vertical" id="apiList" sessionName="listOfAPIs" nextPage="apiDetails"
cardpanel="example2">
        <st:layout>
          <st:field name="optionText" displayInList="true"/>
        </st:layout>
      </st:list>
    </st:panel>
    <st:panel id="example3" title="Example 3" page="example3" />
  </st:tabPanel>
</st:panel>

```

Next, add the *EWD.sencha.onListBackBtnTapped()* function into your *stdemo.js* file:

```

EWD.sencha.onListBackBtnTapped = function() {
  Ext.getCmp("example2").setActiveItem(0, {type:"slide", direction: 'right'});
  Ext.getCmp("apiDetails").destroy();
  Ext.getCmp("listBackBtn").hide();
};

```

So this will slide back the API List back into view inside the *example2* Card Panel, it destroys the injected instance of the *apiDetails* panel and hides the Back button again.

Finally we need to ensure that the Back button appears when *apiDetails.ewd* slides into view, so edit *apiDetails.ewd*:

```

<ewd:config isFirstPage="false" pageType="ajax">

<st:js at="end">
  Ext.getCmp("listBackBtn").show();
</st:js>

<st:panel id="apiDetails" html="List item was selected!" />

```

Save, recompile and re-run the application. Now you should see the Back button appearing when you tap an API name and the *apiDetails* panel slides into view, and tapping the Back button should return you to the API list and the button should disappear again.

Note: you could also have used the even simpler *type="autoback"* mechanism that we described earlier. You might want to give it a try!

It should be clear by now that there is one remaining step that we need to understand in this exercise to make it a really useful piece of functionality: how can we make the *apiDetails.ewd* fragment recognize which API we tapped in the List so that it can provide the user with information specific to that particular one?

Actually it's very simple. By using the *nextPage* attribute in the `<st:list>` tag, EWD is already passing the number of the item you tapped. This number equates to the number in the option array that you created in the pre-page script. So if the user taps the first item, EWD returns 1.

EWD returns the item number as a name/value pair that it automatically adds to the URL that is used to fetch the *apiDetails* fragment. For example, if I was using WebLink, the request for the *apiDetails* fragment would look something like this:

```
http://192.168.1.110/scripts/mgwms32.dll?  
MGWCHD=0&MGWAPP=ewdwl&app=stdemo&page=apiDetails&ewd_token=MWCFjJQi9sShTZIE  
N1iXkYI2d3K55&n=v8eU35uZWEIWPVZKZMQ3eKJpgvHp30&listItemNo=1&ewdrn=612485987
```

Highlighted in red is the name/value pair that EWD added when I tapped the first API in the List.

The trick to using this is to add a pre-page script to *apiDetails.ewd*:

```
<ewd:config isFirstPage="false" pageType="ajax" prepagscript="getAPIDetails^stdemo">  
<st:js at="end">  
  Ext.getCmp("listBackBtn").show();  
</st:js>  
<st:panel id="apiDetails" html="List item was selected!" />
```

In this pre-page script, we can pick up the value of that name/value pair using the EWD API *getRequestValue()*.

However, we have a bit of a problem: all that EWD is returning is the number of the selected option, but that was just a number we used to sequence the list of APIs in the menu: it doesn't provide a direct pointer back to the `^%zewd("documentation", "DOM")` global which is where the API details originated and are held.

We can resolve that by adding an extra couple of lines to the pre-page script that created the List of APIs. Edit that function in your *^stdemo* routine:

```

getAPIList(sessid)
;
n list,name,no
;
s name="",no=0
f s name=${o(^%zewd("documentation","DOM","method",name)) q:name="" d
. s no=no+1
. s list(no,"optionText")=name
;
d deleteFromSession^%zewdAPI("list",sessid)
d mergeArrayToSession^%zewdAPI(.list,"list",sessid)
d saveListToSession^%zewdSTAPI(.list,"listOfAPIs",sessid)
;
QUIT ""

```

What this will do is to persist a copy of the options array in the user's session and then save it as an EWD Session Array named *list*. Note that the Session array is first cleared down in case it already had anything in it. In this example this step isn't strictly required, but it's a good practice to always adopt when using the *mergeArrayToSession* API.

Now we can add the pre-page script for our *apiDetails* page to *^stdemo* as follows:

```

getAPIDetails(sessid)
;
n list,no,name
;
s no=${getRequestValue^%zewdAPI("listItemNo",sessid)}
d mergeArrayFromSession^%zewdAPI(.list,"list",sessid)
s name=${g(list(no,"optionText"))}
d setSessionValue^%zewdAPI("apiName",name,sessid)
;
QUIT ""
;

```

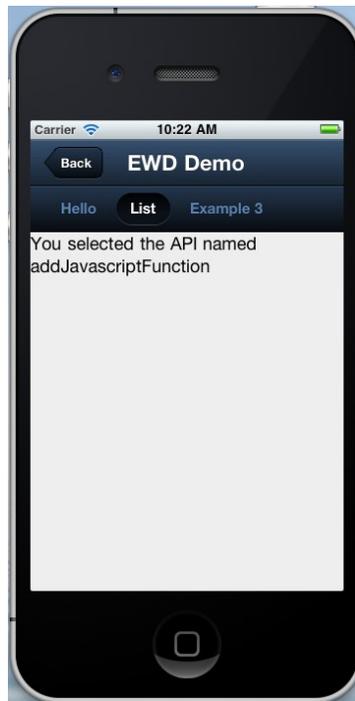
Lets examine what this function will do:

- It gets the value of the *listItemNo* name/value pair using EWD's *getRequestValue()* API
- It then retrieves the original list of options from the EWD Session, by using the *mergeArrayFromSession()* method (this basically reverses the action of the *mergeArrayToSession()* method that was used in *getAPIList()*)
- Now it can retrieve the name for the selected API from the options array by using the list number that was sent by EWD
- We then save that name as an EWD Session value named *apiName*. We can then refer to that name in the fragment's markup.

One last step: we want to display the value of the Session variable (*apiName*) that was created in the pre-page script function.. Edit *apiDetails.ewd*:

```
<ewd:config isFirstPage="false" pageType="ajax" prepagescript="getAPIDetails^stdemo">
<st:panel id="apiDetails">
  <div>
    You selected the API named <?=#apiName ?>
  </div>
</st:panel>
```

Save all the files, recompile the *stdemo* application and run it again. Now when you tap an API in the List, the *apiDetails* panel that slides into view should confirm the one you tapped:



We now have a simple, but fully-functional menu that is driven by data held in GT.M or Caché!

Let's just add one final enhancement: we'll retrieve the purpose of the selected API from the *^%zewd* Global and display it in the *apiDetails* panel. Simply edit the *getAPIDetails()* function in the *^stdemo* routine:

```

getAPIDetails(sessid)
;
n desc,list,no,name
;
s no=$$getRequestValue^%zewdAPI("listItemNo",sessid)
d mergeArrayFromSession^%zewdAPI(.list,"list",sessid)
s name=$g(list(no,"optionText"))
d setSessionValue^%zewdAPI("apiName",name,sessid)
s desc=$g(^%zewd("documentation","DOM","method",name,"purpose"))
d setSessionValue^%zewdAPI("apiPurpose",desc,sessid)
;
QUIT ""
;

```

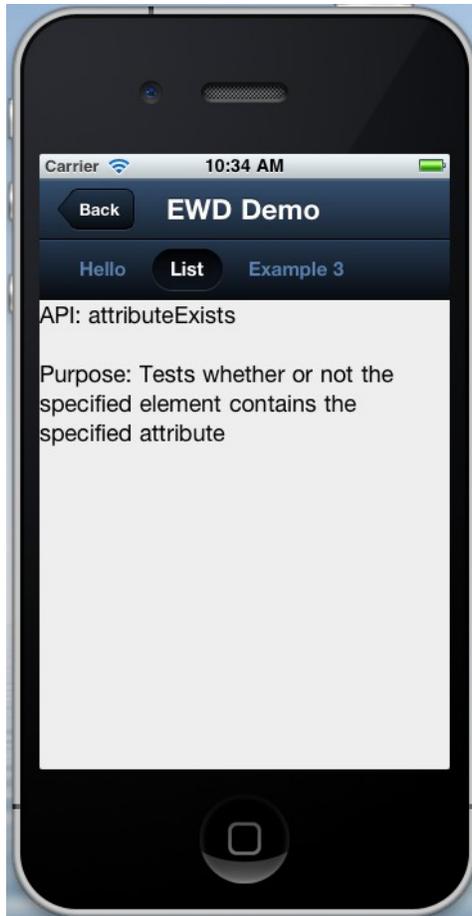
Now edit the apiDetails.ewd page:

```

<ewd:config isFirstPage="false" pageType="ajax" prepagescript="getAPIDetails^stdemo">
<st:panel id="apiDetails">
  <div>
    <div>
      API: <?=#apiName ?>
    </div>
    <br />
    <div>
      Purpose: <?=#apiPurpose ?>
    </div>
  </div>
</st:panel>

```

Save, recompile and re-run the application. Finally we have a proper and useful application that retrieves real details from your GT.M or Cache database in response to the user selecting from a menu:



You now have enough knowledge to begin building mobile applications that can allow a user to navigate through lists of data and then retrieve and display related information. Try modifying the examples to use and display some of your own data.

That completes Part 2 of our EWD/Sencha Touch Tutorial. In Part 3 we'll introduce more widgets including forms and we'll take a closer look at the mechanics of the EWD Session.