EWD

Using the Sencha Touch Custom Tags for Mobile Applications

Tutorial: Part 3

Build 852

Background

This is the third 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) Parts 1 and 2 before embarking on the exercises in this third 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 Part 2 we looked at the List and TabPanel widgets.

In this part we'll look at styling text, modal/pop-up panels and forms. 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 3 you'll need ***build 852 or later*** *of EWD, as some of the features and techniques described in this tutorial have been added since the builds originally released for Parts 1 and 2.*

Lesson 10: Styling Text within Sencha Touch panels

By the end of Lesson 9 in Part 2, we were able to display a List of EWD's DOM APIs, and when one was tapped, a new panel – *apiDetails.ewd* – would slide into view, providing further information about the API.

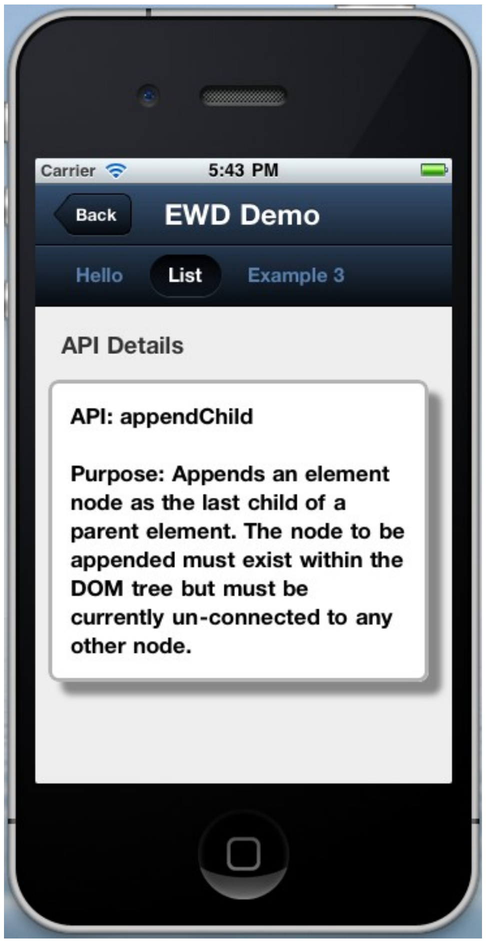
The styling of the text that was displayed in the *apiDetails* panel was very basic, and in this lesson we'll look at a few EWD-specific styling tags that have been provided to give a nice look at feel to the presentation of such text.

The main styling tag is the <*st:pageItem*> tag. This will create a rectangular area with rounded corners in which your markup can be displayed. Let's modify *apiDetails.ewd* to use it.

To begin with, simply place our text markup inside an instance of an <*st:pageItem*> tag, and add a *title* attribute to it, for example:

```
<ewd:config isFirstPage="false" pageType="ajax" prepagescript="getAPIDetails^stdemo">

<st:panel id="apiDetails">

  <st:pageItem title="API Details">
   <div>
     API: <?= #apiName ?>
   </div>
   <br />
   <div>
     Purpose: <?= #apiPurpose ?>
   </div>
  </st:pageItem>

</st:panel>
```

Save this file, recompile and rerun the application. Select an API and you should now see our textual information nicely displayed as follows:



API Details

API: appendChild

Purpose: Appends an element node as the last child of a parent element. The node to be appended must exist within the DOM tree but must be currently un-connected to any other node.

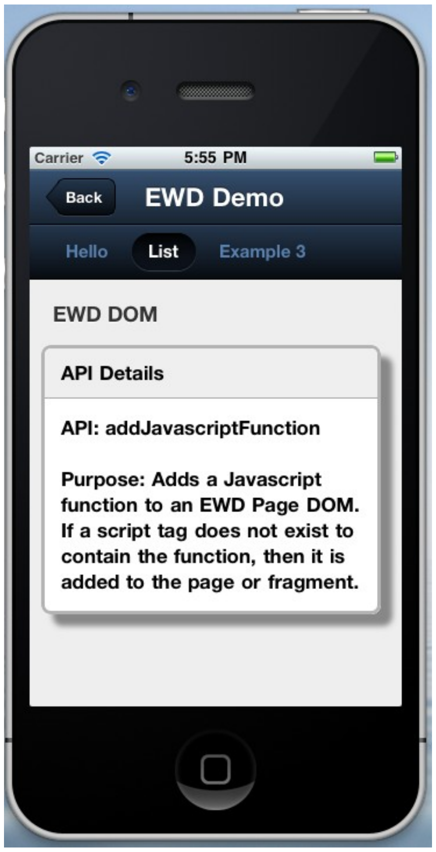
You can turn off the default panel shadowing by simply adding the attribute *shadow="false":*

```
<st:pageItem title="API Details" shadow="false">
  <div>
    API: <?= #apiName ?>
  </div>
  <br />
  <div>
    Purpose: <?= #apiPurpose ?>
  </div>
</st:pageItem>
```

You can also add a heading to the panel:

```
<st:pageItem title="EWD DOM" header=”API Details”>
  <div>
    API: <?= #apiName ?>
  </div>
  <br />
  <div>
    Purpose: <?= #apiPurpose ?>
  </div>
</st:pageItem>
```

The *apiDetails* panel will now look like this:



EWD DOM

API Details

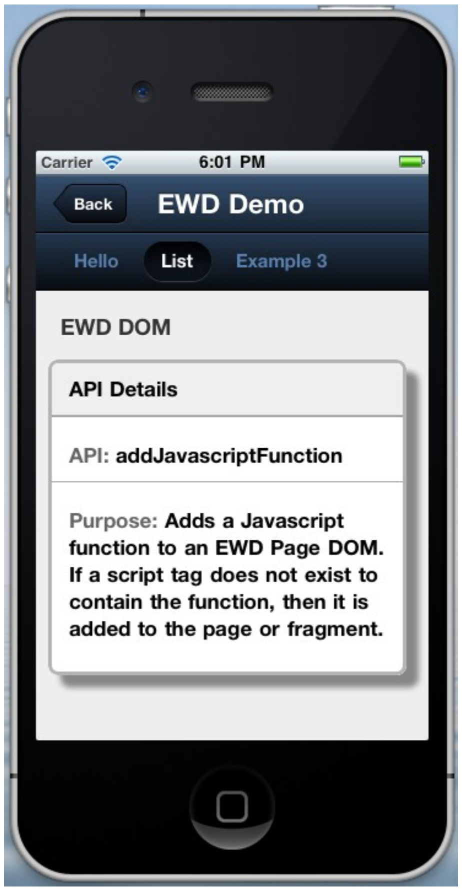
API: addJavascriptFunction

Purpose: Adds a Javascript function to an EWD Page DOM. If a script tag does not exist to contain the function, then it is added to the page or fragment.

We can further enhance the display by using *<st:pageItemField>* tags for the API and Purpose values instead of the raw *<div>* tags that we originally used:

```
<st:pageItem title="EWD DOM" header=”API Details”>
  <st:pageItemField label="API: " value="#apiName" />
  <st:pageItemField label="Purpose: " value="#apiPurpose" />
</st:pageItem>
```

Now the *apiDetails* panel will look like this:



EWD DOM

API Details

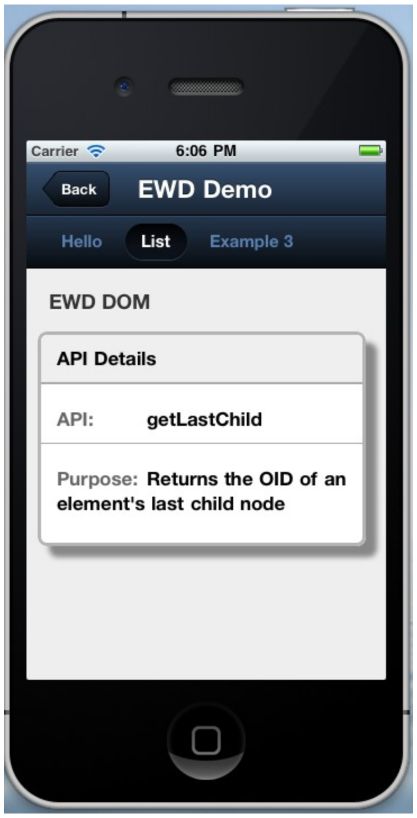
API: addJavascriptFunction

Purpose: Adds a Javascript function to an EWD Page DOM. If a script tag does not exist to contain the function, then it is added to the page or fragment.

We can make one final enhancement, and allocate a width for the label:

```
<st:pageItem title="EWD DOM" header="API Details" labelWidth="75">

   <st:pageItemField label="API: " value="#apiName" />
   <st:pageItemField label="Purpose: " value="#apiPurpose" />

 </st:pageItem>
```

The *apiDetails* panel now looks as follows:



EWD DOM

API Details

API: getLastChild

Purpose: Returns the OID of an element's last child node

You can have as many <*st:pageItemField*> tags as you like inside a single <*st:pageItem*> tag, and an <*st:panel*> tag can contain as many <*st:pageItem*> tags as you want. You can even next <*st:pageItem*> tags inside each other: try it and see what happens!

Lesson 11: Modal Panels

Another way of presenting information to the user is to use modal pop-up panels. Sencha Touch makes this really easy to do and the appearance is very slick.

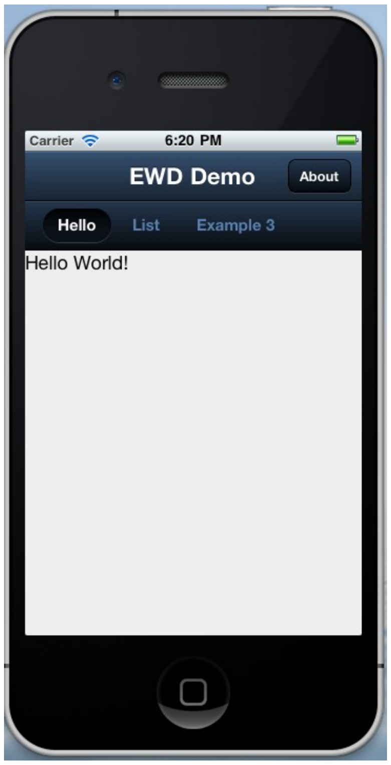
A very common use of pop-up panels in mobile applications is in association with toolbar buttons. Clicking a button can trigger the appearance of a modal panel, and you can even get the panel to line itself up visually with the button the panel is associated with. You'll see this visual effect used a lot in *iPad* applications. You can now do the same thing using EWD and Sencha Touch.

So let's try it out. What we're going to do is to first add an *"About"* button that will be always visible on the right hand side of the top toolbar. To do this, edit the *tabPanel.ewd* page:

```
<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 type="autoback" id="listBackBtn" text="Back" hidden="true" />
    <st:spacer />
    <st:toolbarButton id="aboutBtn" text="About" />
  </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>
```

Not the *<st:spacer>* tag which we use to force the *About* button over to the right-hand side of the toolbar.

Save, recompile and re-run and the application will look like this when it starts:



Now we'll add the modal panel to the *tabPanel.ewd* page, and also a handler for the *About* button:

```
<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 type="autoback" id="listBackBtn" text="Back" hidden="true" />
    <st:spacer />
    <st:toolbarButton id="aboutBtn" text="About" handler="EWD.sencha.onAboutBtnTapped"/
>
  </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>

<st:panel id="aboutPanel" floating="true" width="250" height="200" modal="true"
hidden="true">
    <div>
     About EWD!
    </div>
</st:panel>
```

Notice that we've put the modal panel **outside** the main panel. This is important to ensure that it behaves correctly.

The key attributes that turns a panel into a modal pop-up one are:

- *floating =" true"*
- *modal = "true"*

We're also specifying the size of the panel using the *width* and *height* attributes, and initially the panel should be hidden.

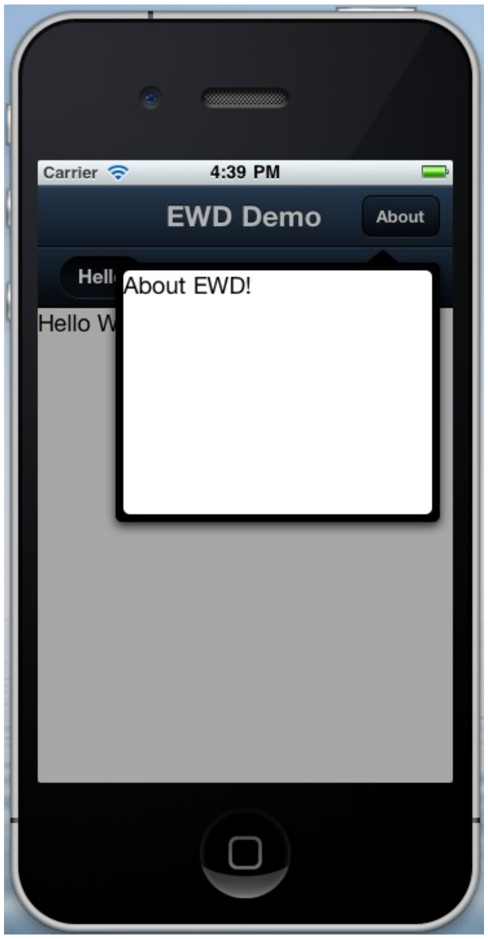
Initially we're just putting a simple bit of text inside the panel. Once we have it working we'll make it more interesting.

Now we need to define that handler for the *About* button, so add the following to your *stdemo.js* file:

```
EWD.sencha.onAboutBtnTapped = function(btn) {
  Ext.getCmp("aboutPanel").showBy(btn);
};
```

The idea, of course, is to make the modal panel appear when the user taps the *About* button. Notice that we're using a different method this time: *showBy()* rather than *show()*. This is a really cool method that positions the modal panel relative to another component. Note we're passing in a parameter – *btn*. That is a pointer to our *About* button and by using that in the showBy() method, the modal panel should line itself up relative to the button.

So, save the tabPanel.ewd page and the stdemo.js file and recompile the stdemo application. Now when you run it and you click the About button, you should see the following:

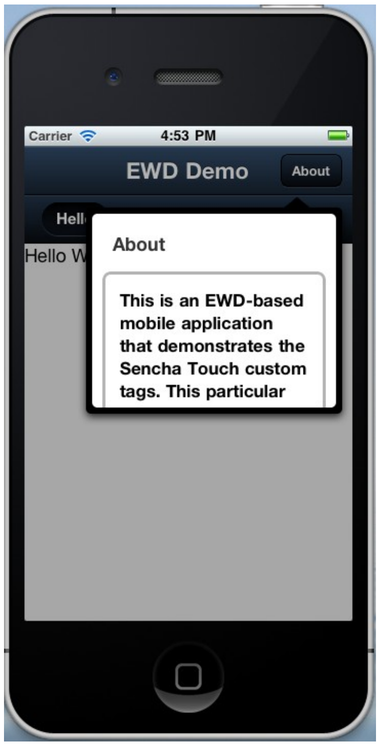


Tap anywhere outside the panel and it it will close automatically and you'll get access to the main panels again.

Now the really nice thing about modal panels is that they can hold pretty much anything that a normal docked panel can hold, eg scrolling text , forms and lists. So, for example, let's put some more meaningful text in that panel and we'll use the <*st:pageItem*> tag to format it:

```
<st:panel id="aboutPanel" floating="true" width="250" height="200" modal="true" hidden="true"
scroll="vertical">
  <st:pageItem title="About" shadow="false">
    <div>
     This is an EWD-based mobile application that demonstrates the Sencha Touch
custom tags.  This particular
     modal pop-up panel was described in Part 3 of the tutorial.
    </div>
  </st:pageItem>
</st:panel>
```

We've put quite a bit of text in it, and we could have made the panel taller by increasing the height. However, just to demonstrate the capabilities, we've told the modal panel to be scrollable in a vertical direction. Save the modified tabPanel.ewd file, recompile it and re-run the application. This time you should see:



This is an EWD-based mobile application that demonstrates the Sencha Touch custom tags. This particular

Try scrolling the text in the panel by touching and swiping it.

You'll soon find that when you use modal panels, it will be difficult to find a particular height and width that will work satisfactorily on both a tablet (eg an iPad) and a phone. Ideally you'll want to use a larger panel on a tablet since you have the available space, but restrict the size on the phone. Indeed you may even decide not to use modal panels if the user is using a phone.

One of the nice things about Sencha Touch is that you can determine the type of device the application is running on really easily, using the property:

```
Ext.is.Phone
```

This will return true if the user is using an iPhone or Android phone, but false if the user is using anything else (including a compatible desktop browser such as Chrome).

So, we could adjust the size of the modal panel dynamically in the *About* button handler as follows:

```
EWD.sencha.onAboutBtnTapped = function(btn) {
  var aboutPanel = Ext.getCmp("aboutPanel");
  if (Ext.is.Phone) {
    aboutPanel.setHeight(200);
    aboutPanel.setWidth(250);
  }
  else {
    aboutPanel.setHeight(500);
    aboutPanel.setWidth(400);
  }
  Ext.getCmp("aboutPanel").showBy(btn);
};
```

Try it and see!

Lesson 12: Using Forms

So far we've just looked at UI functionality that allows the retrieval and display of data from your database. Of course you'll often want to build mobile applications that allow the user to save and update information in your database.

In order to do that, you'll need to be able to create and use forms within your mobile application.

First, however, a word of caution: mobile applications are all about slick, fast user interfaces. As soon as you present a user with a form field where they need to type something in, you're stopping them in their tracks. Although some text entry is inevitable and unavoidable in most business applications, the trick is to keep this to an absolute minimum. There are often ways of getting the user to enter meaningful and useful data without requiring them to type text, for example by using menus, radio buttons, checkboxes, sliders, etc.

With that in mind, let's look at how to present a form to a user.

We'll start with a simple but useful form – a user login form. This will allow you to ensure that only authorized users can access the application. Usually the login form will be the first thing that comes up when the application is started, and that's what we'll now do with our demo application.

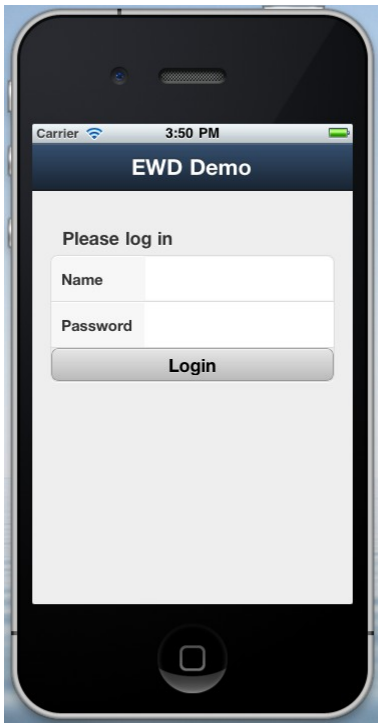
So the first thing is to create a new EWD file named *login.ewd* in your *stdemo* application directory (eg *c:\ewdapps\stdemo\login.ewd* or */ewd/ewdapps/stdemo/login.ewd*) containing the following:

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

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

    <st:form id="loginForm" submitOnAction="false">
      <st:fieldset title="Please log in">
        <st:field type="text" id="name" label="Name" labelWidth="33%" />
        <st:field type="password" id="password" label="Password" labelWidth="33%" />
        <st:field type="submit" text="Login" style="drastic_round" />
      </st:fieldset>
    </st:form>

</st:panel>
```

We're going to make this the main content page – ie what appears when the application is started, so we need to edit the *index.ewd* page:

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

<st:container rootPath="/sencha-1.0/" contentPage="login" 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>
```

Save these two files and recompile the *stdemo* application. When you restart it, you should see:



You'll find that this form won't do anything yet. Let's fist analyse what we specified to make this form appear:

- We've made this *login.ewd* page the main content page, so it now contains the main outer panel and the toolbar. We haven't specifed any buttons in it yet
- We added a form panel, denoted by the *<st:form>* tag

- Inside the <*st:form*> we specified an <*st:fieldset*> tag which creates the styled grouped set of form fields together with its title ("*Please log in*")
- We then added three <*st:field*> tags for the username field, password field and Login button. Note the *id* we gave the username and password fields: these will be important later. The shape of the submit button is determined by the style attribute. *drastic-round* is one of the standard Sencha Touch button styles and a good one for forms. Try running the Sencha Touch *Kitchen Sink* demo application to see many of the others that are available.

So that's our basic form displaying itself properly. Next we'll animate it so that when the *Login* button is clicked, our tab panel slides into view. Initially we'll not worry about validating the username and password.

There are several things we need to do in order to animate the form. Remember from the earlier exercises that animating a set of panels requires a *card panel* as a container, so we'll add that first, with the form initially the only card inside it:

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

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

  <st:cardpanel id="mainCardPanel">
    <st:form id="loginForm" submitOnAction="false">
      <st:fieldset title="Please log in">
        <st:field type="text" id="name" label="Name" labelWidth="33%" />
        <st:field type="password" id="password" label="Password" labelWidth="33%" />
        <st:field type="submit" text="Login" style="drastic_round" />
      </st:fieldset>
    </st:form>
  </st:cardpanel>

</st:panel>
```

Next we add a couple of attributes to the submit button field to tell it the name of the fragment we want to fetch when it is clicked, and the id of the cardpanel to use as the container for the animation which is *slide* by default:

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

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

  <st:cardpanel id="mainCardPanel">
    <st:form id="loginForm" submitOnAction="false">
      <st:fieldset title="Please log in">
        <st:field type="text" id="name" label="Name" labelWidth="33%" />
        <st:field type="password" id="password" label="Password" labelWidth="33%" />
        <st:field type="submit" text="Login" style="drastic_round" nextpage="tabPanel"
cardpanel="mainCardPanel" />
      </st:fieldset>
    </st:form>
  </st:cardpanel>

</st:panel>
```

Now before we try running the application again, we'll have to do some surgery on the *tabPanel.ewd* page. That's because it was originally our main page and contains a full-screen panel, toolbar, buttons etc. All that stuff is now being provided by our new *login.ewd* page, so we need to take it out of the *tabPanel.ewd* page and just let it contain the tab panel alone. Edit it to look as follows (take a copy of the original file first because there is stuff in it that we'll want to copy later!):

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

  <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>
```

Now try saving the edited files, recompile and re-run the application. You should find that when you click the submit button, it will slide the tab panel into view, and everything should then work as before – it's really starting to look like a proper application now!

Well, except that we've lost the back button and the about button. So let's get them reinstated. From your copy of *tabPanel.ewd*, copy the toolbar and button and replace the mimimal toolbar we'd put into *login.ewd*:

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

<st:panel id="mainPanel" fullscreen="true" layout="card">
 <st:toolbar id="topToolbar" dock="top" title="EWD Demo">
   <st:toolbarButton type="autoback" cardpanel="example2" id="listBackBtn"
text="Back" hidden="true" />
   <st:spacer />
   <st:toolbarButton id="aboutBtn" text="About"
handler="EWD.sencha.onAboutBtnTapped" />
 </st:toolbar>

 <st:cardpanel id="mainCardPanel">
   <st:form id="loginForm" submitOnAction="false">
     <st:fieldset title="Please log in">
       <st:field type="text" id="name" label="Name" labelWidth="33%" />
       <st:field type="password" id="password" label="Password" labelWidth="33%" />
       <st:field type="submit" text="Login" style="drastic_round" nextpage="tabPanel"
cardpanel="mainCardPanel" />
     </st:fieldset>
   </st:form>
 </st:cardpanel>

</st:panel>
```

Notice also an extra attribute you should add to the back button: *cardpanel="example2"*. That tells EWD that this back button is for use by the cardpanel with an *id* of *example2*. If you look in *tabPanel.ewd* you'll see that's the card panel used by the List of APIs and its associated animation.

Just one more thing to do: we need to put the modal panel back. Copy it from your copy of *tabPanel.ewd* and place it at the bottom of *login.ewd*. So the completed *login.ewd* page should now look like this:

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

<st:panel id="mainPanel" fullscreen="true" layout="card">
  <st:toolbar id="topToolbar" dock="top" title="EWD Demo">
    <st:toolbarButton type="autoback" cardpanel="example2" id="listBackBtn" text="Back"
hidden="true" />
    <st:spacer />
    <st:toolbarButton id="aboutBtn" text="About" handler="EWD.sencha.onAboutBtnTapped" />
  </st:toolbar>

  <st:cardpanel id="mainCardPanel">
     <st:form id="loginForm" submitOnAction="false">
       <st:fieldset title="Please log in">
         <st:field type="text" id="name" label="Name" labelWidth="33%" />
         <st:field type="password" id="password" label="Password" labelWidth="33%" />
         <st:field type="submit" text="Login" style="drastic_round" nextpage="tabPanel"
cardpanel="mainCardPanel" />
       </st:fieldset>
     </st:form>
  </st:cardpanel>

</st:panel>

<st:panel id="aboutPanel" floating="true" width="250" height="200" modal="true"
hidden="true" scroll="vertical">
  <st:pageItem title="About" shadow="false">
    <div>
     This is an EWD-based mobile application that demonstrates the Sencha Touch
custom tags.  This particular
     modal pop-up panel was described in Part 3 of the tutorial.
    </div>
  </st:pageItem>
</st:panel>
```

Save this version of *login.ewd*, recompile it and try running the application. Now everything should be back as it was, except the user has to complete a login form first.

Notice that after the submit button is pressed, there is no way to bring back the form which is precisely the behaviour was want.

Now let's look at how we validate the username and password. Everything we need to do will take place in the pre-page script of *tapPanel.ewd* since this is the *"nextPage"* that we're requesting when the submit button is clicked. We already have a pre-page script for that page, which is currently just fetching the list of DOM APIs. Look in the routine file (*stdemo.m* or open *stdemo.mac* using Caché Studio) and you'll see that it currently contains:

```
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 ""
```

Add the lines indicated below in this edited version:

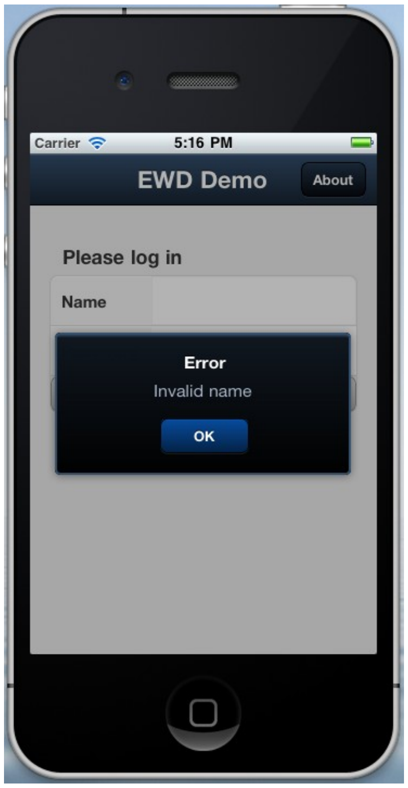
```
getAPIList(sessid)
 ;
 n list,name,no,password
 ;
 s name=$$getRequestValue^%zewdAPI("name",sessid)
 s password=$$getRequestValue^%zewdAPI("password",sessid)
 i name'="rob" QUIT "Invalid name"
 i password'=1234 QUIT "Invalid password"
 d setSessionValue^%zewdAPI("username",name,sessid)
 ;
 s name="",no=0
 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 ""
```

Save and recompile the *stdemo* routine. On GT.M it's also a good idea to run the following in the GT.M shell:

```
do relink^%zewdGTM
```

This ensures that the back-end *m_apache* processes relink their copy of the *stdemo* routine so they're using the latest version.

Now try running the application and try just clicking the submit button without entering any name or password. This time you won't be let in and an alert will up:



In fact you won't be able to get any further unless you enter a username of ***rob*** and a password of ***1234***, which is the validation rules we'd hard-coded into the pre-page script. Enter the correct username and password, however, and the tab panel slides into view.

Note how simple EWD makes form field validation:

- We pick up the submitted values using EWD's *$$getRequestValue()* method. The field name you use is the same as the *id* attribute you specified in the form, ie:

```
        <st:field type="text" id="name" label="Name" labelWidth="33%" />
        <st:field type="password" id="password" label="Password" labelWidth="33%" /
>


        s name=$$getRequestValue^%zewdAPI("name",sessid)
        s password=$$getRequestValue^%zewdAPI("password",sessid)
```

- We’ve just used some simple hard-coding to validate the username and password, but you can use whatever user authentication functionality you already have in place on your GT.M or Cache system, or write your own for your new application..

- In order to signal a validation failure to EWD, simply QUIT with a non-empty string return value. Whatever string you specify will automatically be used as the text in a Sencha Touch alert panel that will pop up.

- Otherwise your pre-page script should continue on its way and ultimately QUIT will an empty string as its return value. This signals to EWD that the pre-page script ran without any validation errors, so it can now safely render the contents of the *nextpage* fragment.

You now have a working username/password form! Try playing about with the validation logic in the pre-page script, add new usernames, hook it up to an authentication global etc.

Lesson 13: The EWD Session

You may have noticed that there was an extra line we added to the username/password validation code that wasn't described in detail:

```
getAPIList(sessid)
 ;
 n list,name,no,password
 ;
 s name=$$getRequestValue^%zewdAPI("name",sessid)
 s password=$$getRequestValue^%zewdAPI("password",sessid)
 i name'="rob" QUIT "Invalid name"
 i password'=1234 QUIT "Invalid password"
 d setSessionValue^%zewdAPI("username",name,sessid)
 ;
 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 ""
```

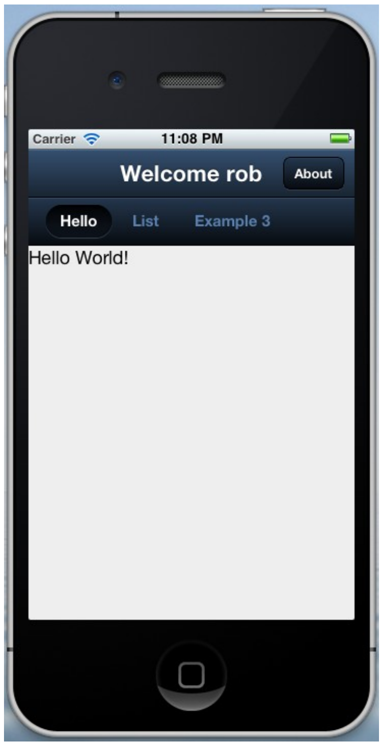
We've seen this command being used before: it saves a name/value pair into the user's persistent EWD Session storage. In the example above, we're persisting the user's username as a an EWD Session variable named *username*. By doing this, any fragment that is fetched by the user's instance of the application can access this variable.

We can demonstrate this in the tab panel page which appears if the user successfully logs in. Let's change the toolbar title to give a personalized greeting. Edit *tabPanel.ewd* as follows:

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

<st:js at="end">
 Ext.getCmp("topToolbar").setTitle('Welcome <?= #username ?>');
</st:js>

 <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>
```

The Sencha Touch function *setTitle()* will dynamically modify the tootbar's title. Notice how we're referencing the EWD Session Name within the Javascript: *<?= #username ?*
>

Save this *tabPanel.ewd* file and recompile it. Now when you log in, you'll see the following:



Hello World!

We've used the EWD Session in a number of ways so far without really studying what's been going on. In fact, the Session is one of the key parts of EWD, and understanding how it works is important if you're going to get the full benefit of EWD. So now we'll delve deep into what's going on behind the scenes in EWD.

User Sessions and EWD Session Storage

In fact let's start right back the URL you use to start an EWD application. In the case of our *stdemo* application, in order to start it up you'll have used a URL similar to one of the following (your IP address or domain name will be different of course):

- **GT.M**: http://192.168.1.100/ewd/stdemo/index.ewd
- **CSP**: http://192.168.1.100/csp/ewd/stdemo/index.csp
- **WebLink**: http://192.168.1.100/scripts/mgwms32.dll?MGWLPN=LOCAL&MGWAPP=ewdwl&app=stdemo&page=index

In all three cases, what we're doing is requesting the page named *index.ewd* from the EWD application named *stdemo*. We are allowed to do this with a simple "static" URL because *index.ewd* was configured to be a "First Page" by virtue of its <*ewd:config*> tag:

<ewd:config isFirstPage="true">

What happens when you invoke a FirstPage with an untokenised URL is that EWD creates a new, unique Session ID. This is simply an integer value but it is unique to you, the user. Each time you subsequently click a link or button in an EWD page within the application you've started, you send a URL back to the Caché server that identifies your Session ID via a set of randomly-generated name/value pairs (or tokens) that will have been automatically added to the URL by EWD.

Actually those tokens do two things:

- They identify you as a *bona fide* user who is allowed to request the fragment specified in the URL from the Caché/GT.M back end system. If the token is missing or not recognized, EWD will return an error fragment or alert back to the browser and refuse to retrieve the fragment

- They securely identify your unique session Id. This is achieved via a lookup table (mapping tokens to session Ids) that is automatically maintained by EWD within the GT.M or Caché back-end system. EWD never directly exposes the user's session Id within the HTML or Javascript that runs in the browser. Instead it uses the random tokens on the URLs as a highly secure proxy to the session Id. If the user had to log in to access an EWD application, you (the owner of the application) can then be highly confident that once the user has logged in, subsequent requests for fragments that make up that application can only have come from that user and not from someone simply pretending to be that person. It is this high degree of security that has made EWD a popular framework for applications that involve access to sensitive healthcare and financial data.

If the EWD page that has been requested has an associated pre-page script it is invoked before anything else happens. the user's unique session Id (*sessid)* is passed into the method by EWD. That's why your pre-page script functions must always require *sessid* as their one and only input parameter.

Now that the pre-page script knows the user's unique session Id, calls can be made to the EWD Session API methods (eg *getSessionValue*, *setSessionValue* etc) to access and manipulate the user's specific Session storage. These APIs are described in detail later.

EWD's Session Storage (known simply as the *EWD Session*) provides a means by which you, the developer, can retain user-specific data for the lifetime of the user's session. A user's session starts when they invoke the initial untokenised URL to an EWD application's FirstPage, and finishes when one of two things happens:

- Most commonly, after the user is inactive for a predetermined duration (20 minutes by default), at which point their browser either:
    - Automatically refreshes, forcing a reload of the index (or whatever their first page was), which starts a new session (this is the default EWD/Sencha Touch action); or
    - Redirects them to a static URL or the first page of a different EWD application (something that you, the developer can decide to do)
- Less frequently, the user invokes an action (that you, the developer, has provided them with) that explicitly terminates their session and redirects them to a static URL

Once an EWD session times out, it is no longer accessible even with a validly tokenized URL. Any persistent EWD Session Data that is associated with a user session that has timed out is automatically cleared down by EWD's garbage collector.

The EWD Session Storage (aka *EWD Session*) is a key part of the EWD architecture and determines how your EWD pages are designed and how information is conveyed between the browser and back-end database. So let's examine it in some more detail.

Designer versus Programmer

You've seen in this tutorial that an EWD application consists of two distinct parts:

- The EWD pages or fragments
- Back-end methods: ie Pre-page Script functions that run within GT.M or Caché.

These are often referred to as the "front end" and "back end" respectively.

One of the key concepts of EWD is that these two parts of an application can be developed by two different people:

- Front-end design can be carried out by a web application designer
- Back-end method development can be carried out by a programmer who is familiar with GT.M or Caché, the data models that define the organisation's database, and business logic (ie functions, procedures and APIs) that already exist within the organisation

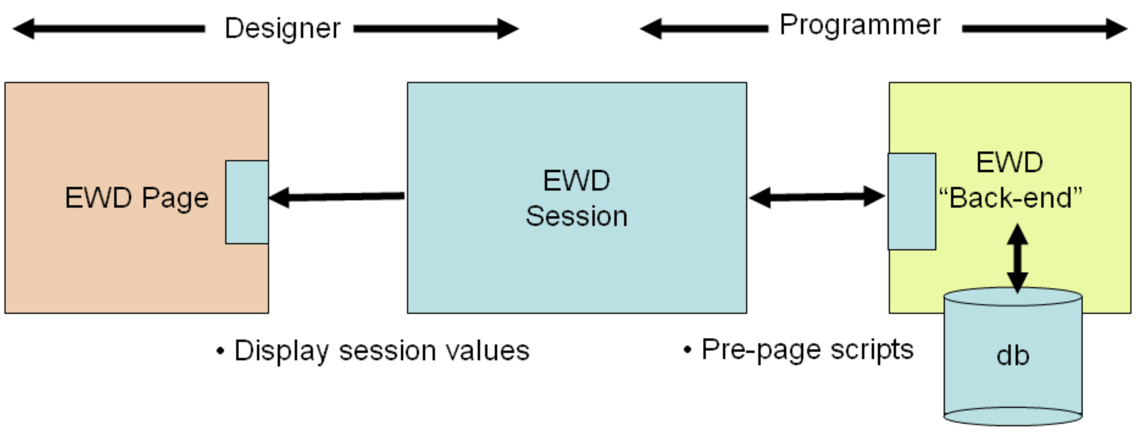
The two halves of an EWD application require different disciplines and knowledge, and EWD allows each half to be developed by the correct person with the correct skills. Indeed the designer of the front-end does not have to know anything about the back-end in order to create his/her part of the application. Similarly, the programmer can develop all the scripts needed by an EWD application without really having to know anything about how or why they are being used.

In practice, of course, many EWD applications will be developed by one or more people who simultaneously adopt both roles of designer and programmer, but nevertheless, EWD provides a very important separation between design and programming.

Unlike probably most other web application development environments, in EWD there is no use of "data binding". In other words, the front-end page page/fragment definitions have no direct access to back-end data. This is deliberate and, contrary to most industry preconceptions, turns out to be highly advantageous.

In fact there's a critical problem with data binding: it irrevocably interlinks the front end design with the back-end data model. Whilst fine for rapid development of quick demonstration applications, for real-world applications it is distinctly bad news for downstream maintenance and change. Let's face it: all real-world applications, no matter how simple and trivial, change over time. With data binding, if the front-end design needs changing, it will almost certainly have an impact on the back-end database. Similarly, changes to the back-end database schema will almost certainly impact on the front-end design logic. The result is a time-consuming, complex and unnecessary mess that consumes hours of development time, and/or hinders or, in the worst cases, completely prevents enhancements to the application.

EWD takes a totally different approach and deliberately separates and de-couples the front-end from the back-end. However, it provides an intermediate boundary layer between the two: the EWD Session which each can access. The diagram below summarises the architecture:

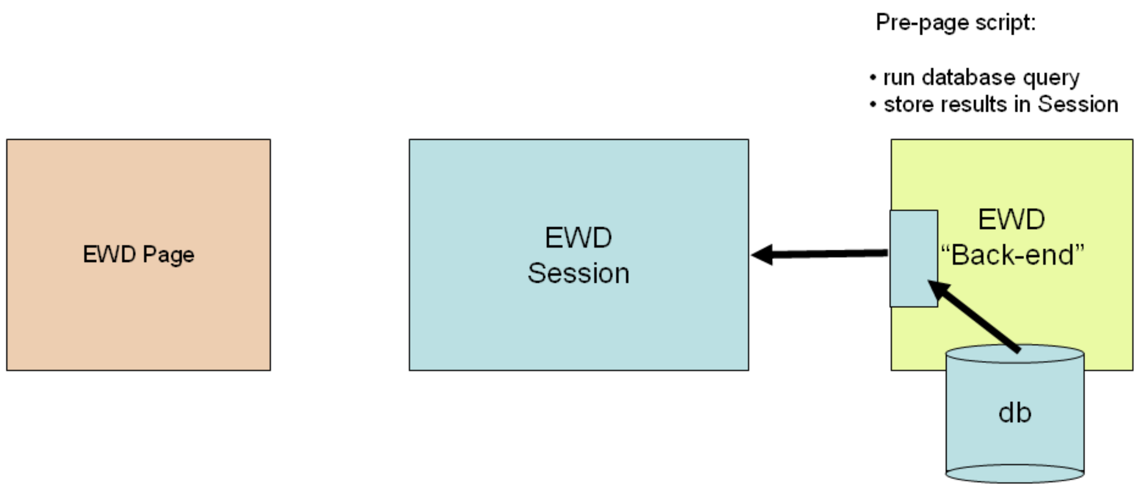


Thus, the front-end designer has access to and can reference and use EWD Session Variables and Arrays in his/her pages. The designer does not need to know where they came from originally: he/she simply needs to know that they must exist in the EWD Session and knows their structure (eg a simple name/value pair or a multi-dimensional array with a particular structure).

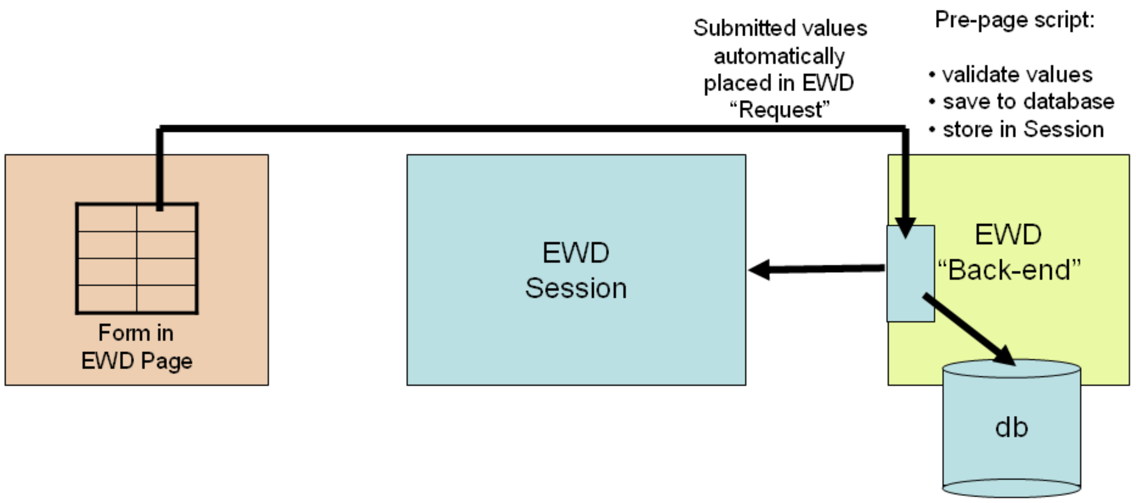
In EWD/Sencha Touch applications, the designer has no direct role in creating EWD Session data.

The task of creating and maintaining the data in the EWD Session is the sole responsibility of the back-end programmer. Indeed the programmer's role can be thought of as simply moving data between the back-end database and the EWD Session:

- His/her Pre-Page Scripts will run queries against the database and transfer the results into EWD Session Variables and/or Arrays (as summarised below);



- When a form has been submitted, his/her pre-page script will pick up the corresponding Request values (EWD automatically puts them there), validate them, and, if they are OK, save them back into the database and perhaps keep a copy in the EWD Session (as summarised below)



The programmer can create all the scripts needed for an application without really having to have any knowledge about the pages/fragments in which they will actually be used.

So, with EWD, the two participants, the designer and programmer, can work together, but separately. The front-end design can be changed significantly without affecting the back-end database, and often without even affecting the pre-page script functions. Conversely, the back-end database can be altered without any impact on the front-end design: provided the new database schema can be mapped in and out of the same EWD Session structures, the front-end will still operate identically because it won't see any change in the EWD Session.

There's another great benefit of this architecture. An EWD application can initially be fully developed using mocked-up, hard-coded data in initial pre-page scripts. Once the front-end is agreed upon, the pre-page script functions can be edited by a database programmer to maintain the original EWD Session data structures but now using the real database and real business logic. The change from design prototype to fully operational system requires no further work by the designer!

There's a further benefit: it is quite possible to migrate an EWD application from one back-end database technology to another. Provided the new pre-page scripts maintain the original EWD Session data structures, the front-end design will neither know nor care that it's being driven by an entirely different back-end database!

Accessing the Session from within an EWD Page

The designer accesses the EWD Session by using references to the data value(s) he/she wants to display. There are three primary techniques:

a) In an attribute of a Sencha Touch Custom tag. The name of the EWD Session variable should be prefixed with a # character, eg

```
<st:panel title="#myTitle" />
```

b) Anywhere else in the content of the page/fragment, including within Javascript, the *<?= #sessionName ?>* syntax is used, eg:

```
<div>The title <?= #myTitle ?> has come from the Session</div>

Ext.getCmp("topToolbar").setTitle('<?= #myTitle ?>');
```

c) by specifying the *sessionName* attribute of a Sencha Touch Custom Tag. Usually in this context the *sessionName* refers to an EWD Session Array through which the EWD Sencha Touch custom tag processor will iterate in order to display its contents, eg:

```
<st:list sessionName="listOfAPIs" />
```

You've seen examples of all three techniques in the preceding examples of this tutorial. Now you hopefully have a better idea what they mean!

EWD also provides more primitive custom tags for iterating through EWD Session Arrays, but in Sencha Touch applications these will be rarely used. Refer to the EWD API Guide (http://gradvs1.mgateway.com/download/APIGuide3.0.618.pdf) for more details if you are interested.

Creating and Manipulating EWD Session Data

The programmer accesses the EWD Session through a range of APIs that EWD provides. Although the EWD Session is physically implemented using Global storage in GT.M and Caché, the programmer should **never** access the Global directly. This is because M/Gateway reserves the right to change the Global structures in future releases of EWD if it becomes necessary. The APIs will always retain their correct behaviour irrespective of such internal changes, whereas direct access to the EWD Session Global by programmers almost certainly won't!

Although EWD provides quite a large range of APIs for Session management, in practice there are only a small number of APIs that the programmer will typically need to use.

All EWD Session APIs include as their last parameter the user's session Id, denoted by the variable *sessid*. The most common APIs are as follows:

a) Creating a simple name/value pair

> *do setSessionValue^%zewdAPI(sessionName,value,sessid)*
>
> This creates a name/value pair named according to the *sessionName* parameter with a value provided by the *value* parameter.

b) Getting the value of a simple name/value pair Session variable

> *set value=$$getSessionValue^%zewdAPI(sessionName,sessid)*
>
> This returns the value of a name/value pair named according to the *sessionName* parameter.

c) Deleting a simple name/value pair Session variable

> *do deleteFromSession^%zewdAPI(sessionName,sessid)*
>
> This deletes a name/value pair named according to the *sessionName* parameter.

d) Creating a Session array

> First create a local array of any structure you like, eg
>
> *set array("a","b")=123*
> *set array("a","b","c")="foo"*
> *set array("a","c")="xyz"*
> *set array("d","a")=999*
> *set array("d","b")="bar"*
>
> Then merge this to a corresponding EWD Session array using:
>
> *do mergeArrayToSession^%zewdAPI(.array,sessionName,sessid)*
>
> Note that array is called by reference, indicated by the period (.) that prefixes it

Because this API uses a non-destructive merge, it is usually good practice to first clear down anything that might exist in the EWD Session array, ie:

do deleteFromSession^%zewdAPI(sessionName,sessid)
do mergeArrayToSession^%zewdAPI(.array,sessionName,sessid)

e) Getting back a Session array to access/manipulate its contents

First do the reverse of step *d*, ie recover the Session array as a local array using (make sure the local array is initially cleared down):

do mergeArrayFromSession^%zewdAPI(.array,sessionName,sessid)

Note that array is called by reference, indicated by the period (.) that prefixes it

If we applied this to the Session Array created in example *d* above, the local array would contain the following:

array("a","b")=123
array("a","b","c")="foo"
array("a","c")="xyz"
array("d","a")=999
array("d","b")="bar"

If you make any changes to the local array, use the *mergeArrayToSession()* API if you need to save the changes back into the EWD Session.

f) Deleting an EWD Session array

do deleteFromSession^%zewdAPI(sessionName,sessid)

This deletes the entire contents of an EWD Session array named according to the *sessionName* parameter

Processing Submitted Forms

In EWD Sencha Touch applications, when a form is submitted, the values of all the form fields are transferred to an ephemeral storage known as the *Request*. The Request contains the values of any name/value pairs that were sent with an HTTP request (eg a request for a fragment). Those name/value pairs will be either:

- Name/value pairs that were added to the URL (ie using the HTTP GET method)

- Form field values (ie using the HTTP POST method)

The Request is cleared down on every HTTP request, so if you need to process or access the incoming values of any name/value pairs, you must do so immediately in the pre-page script of the associated fragment. If you need any of the Request name/value pairs to persist longer, you must transfer them to the EWD Session using the appropriate API as listed in the previous sub-section.

When you submit a form, the value of each form field's *id* attribute is used as the corresponding Request name.

So let's look again at the pre-page script code we used for validating our example username/password form:

```
n name,password
;
s name=$$getRequestValue^%zewdAPI("name",sessid)
s password=$$getRequestValue^%zewdAPI("password",sessid)
i name'="rob" QUIT "Invalid name"
i password'=1234 QUIT "Invalid password"
d setSessionValue^%zewdAPI("username",name,sessid)
```

Here's the corresponding form:

```
    <st:form id="loginForm" submitOnAction="false">
      <st:fieldset title="Please log in">
        <st:field type="text" id="name" label="Name" labelWidth="33%" />
        <st:field type="password" id="password" label="Password" labelWidth="33%" /
>
        <st:field type="submit" text="Login" style="drastic_round" nextpage="tabPanel"
cardpanel="mainCardPanel" />
      </st:fieldset>
    </st:form>
```

So the value of the *name* field ended up in the Request name/value pair named *"name",* as highlighted in blue. The value of the *password* field ended up in the Request name/value paid named *"password",* as indicated in red.

In the pre-page script, we first get hold of the Request values using EWD's *getRequestValue()* API and then validate the values as required. In the example above we want to persist the value of the name, so we've saved it in an EWD Session variable named *username*.

Although the name and password Request values will disappear when the next request for an EWD fragment is sent from the browser, all subsequent fragments and their associated pre-page scripts have access to the *username* as it is now saved in the user's Session storage.

Hence within the *tabPanel.ewd* page we were able to use the username EWD Session value to modify the toolbar title:

```
<st:js at="end">
 Ext.getCmp("topToolbar").setTitle('Welcome <?= #username ?>');
</st:js>
```

This completes Part 3 of the EWD/Sencha Touch tutorial. In Part 4 we'll look at the other form field types that are provided by Sencha Touch and how to use them within EWD. We'll also look at some of the other really cool widget types that are available as EWD Custom Tags.