EWD

Using the Sencha Touch Custom Tags for Mobile Applications

Tutorial

Build 844

Background

The Sencha Touch Custom Tags provide a set of extensions to Enterprise Web Developer (EWD) that allow extremely rapid development of Caché-and GT.M-based Mobile Web Applications that look and behave like Native Applications, running on the iPhone, iPad and Adroid phones and tablets. EWD/Sencha Touch application will also run on desktop WebKit browsers such as Chrome and Safari.

Key benefits of using EWD in conjunction with the Sencha Touch Custom Tags are:

- One application definition will run on all mobile devices that use WebKit browsers, instead of writing separate versions of applications for the iPhone, iPad and Android devices.
- The development and maintenance effort is orders of maginitude less using EWD & the Sencha Touch tags than writing Native Applications for mobile devices
- Distribution of applications and their updates is not dependent on acceptance in an App Store.

This document provides a tutorial to help you learn how to build Mobile Apps using EWD & the Sencha Touch Custom Tags.

Pre-requisites

Before you begin this tutorial, you should install the latest build of EWD (build 844 or later). Details on how to install and configure EWD are provided in Appendix I.

You will also need to install the latest version of the Sencha Touch Javascript framework. You can obtain this from http://www.sencha.com.

This tutorial makes some assumptions about your configuration which are explained in detail in Appendix I. The key assumptions are:

- Your EWD Application root path will be:
    - c:\ewdapps (Windows)
    - /usr/ewdapps (Linux)
- If you are using Cache, you'll be working in the USER namespace
- If you are using GT.M, the instance you'll be using is in /usr/local/gtm/ewd
- Sencha Touch has been installed under your web server's root directory in a subdirectory named */sencha-1.0*

You'll need to adjust the examples appropriately if your configuration is different.

Lesson 1 : Hello World

In time-honoured fashion, let's start this tutorial by creating a simple "Hello World" application that will run as a Mobile application. We'll analyse in detail how this application works and what all the XML means in the next section, but for now just try the following steps and see how little is involved in developing Mobile applications with EWD.

We're going to name this demo application *stdemo*, so you'll be creating and editing files in the directory *c:\ewdapps\stdemo (or /usr/ewdapps/stdemo)*

You can use any text or HTML editor to create and edit your EWD source pages. In this tutorial we'll assume that you are using a simple text editor such as Notepad.

Create a text file named *index.ewd* (ie *c:\ewdapps\stdemo\index.ewd)* that contains the following:

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

<st:container rootPath="/sencha-1.0/" contentPage="helloworld"
title="Hello World">
  <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/kit-
chensink/resources/img/phone_startup.png" />
    <st:image type="icon" src="/sencha-1.0/examples/kitchensink/re-
sources/img/icon.png" addGloss="true" />
  </st:images>
</st:container>
```

Note: keep each tag on a single line – ie don't include line breaks inside a tag

Next, create a second text file name *helloWorld.ewd* (ie *c:\ewdapps\stdemo\helloWorld.ewd)* that contains the following:

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

<st:panel fullscreen="true" html="Hello World!" />
```

OK, let's compile and run this application. Start up a GT.M session or a Caché Terminal session and change to the USER namespace. Type the following (depending on whether you're using GT.M, CSP or WebLink):

CSP: d compileAll^%zewdAPI("Stdemo",,"csp")
WebLink: d compileAll^%zewdAPI("Stdemo",,"wl")
GT.M: d compileAll^%zewdAPI("Stdemo")

Note the two commas if you're using Caché!!

If you've properly installed and configured things, you should have seen something like the following:

```
USER>d compileAll^%zewdAPI("stdemo",,"wl")
c:\ewdapps\stdemo\ewdAjaxError.ewd
Compiling routine : ewdWLstdemoewdajaxerror.INT
Compiling routine : ewdWLstdemoewderror.INT
c:\ewdapps\stdemo\ewdAjaxErrorRedirect.ewd
Compiling routine : ewdWLstdemoewdajaxerrorredirect.INT
c:\ewdapps\stdemo\ewdErrorRedirect.ewd
Compiling routine : ewdWLstdemoewderrorredirect.INT
c:\ewdapps\stdemo\helloworld.ewd
Compiling routine : ewdWLstdemohelloworld.INT
c:\ewdapps\stdemo\index.ewd
Compiling routine : ewdWLstdemoindex.INT

USER>
```

You can now run the application. Start the browser on your Mobile device and enter the URL:

- **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

You'll need to change the IP address to that of your web server.

What you should see, after a short pause while it loads everything up, is:


If you're running in an iPhone, iPod Touch or iPad, you'll notice that the application is running full screen without Safari's URL Location window being visible. However, Safari's toolbar is still visible at the bottom, and if you wanted to restart the application, you'd need to relaunch Safari and tediously re-enter that lengthy URL.

However, there's a further trick you can do on an iPhone or iPad (Android devices have an equivalent procedure). Click that + sign in the middle of the bottom toolbar (on recent versions of iOS it's a rectangle with an arrow emerging from it) and up will pop the following:
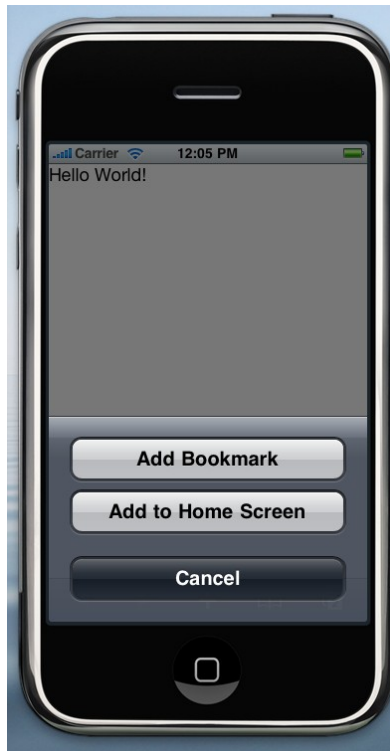
Click the "Add to Home Screen" button and you'll see:
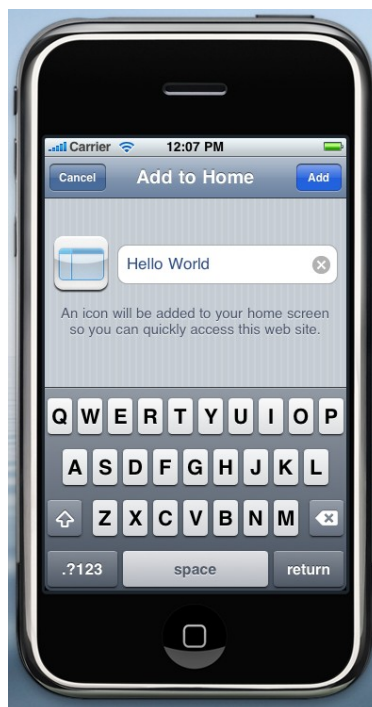
Click the blue Add button in the top right corner, and the Touch-Icon that we'd specified in the index.ewd page's <st:image type='icon'> tag will be added to your iPhone's Home Screen

Now you can start our Demo application by simply touching the icon, just as if it was a Native iPhone/iPad App. What's more, when you start it up this way, you'll see the startup splash screen, and when the application is fully loaded and ready for use, you'll no longer see any trace of Safari's chrome:
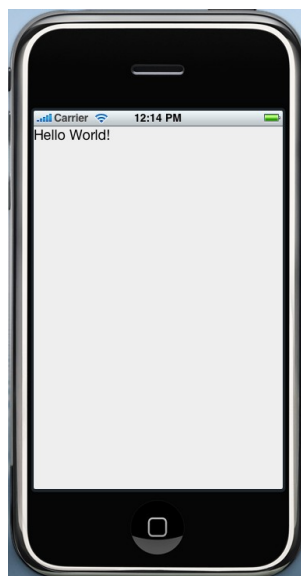
Now it just looks like a Native App!

Lesson 2 : HelloWorld Analysed

Hopefully you'll agree that there wasn't much involved in getting an albeit very simple EWD application up and running. Now let's take a more detailed look at what we did and why we did it.

Every EWD Application requires what is known as the *Container Page* or *First Page*. This is basically an empty HTML page that acts as a container for all the subsequent content that will be sent to the mobile browser by EWD. EWD applications use Ajax techniques for injecting chunks of markup into the container page. As far as your browser is concerned, an EWD application is actually a single page of HTML that never leaves the browser. EWD will handle all that Ajax stuff for you automatically.

The Container Page can be named anything you like (avoid punctuation marks in names). However, by convention, we usually name this page ***index.ewd***. EWD pages must have a file extension of ***.ewd***.

So, our EWD Container page was the file named *index.ewd*. The reason we created it in the path *c:\ewdapps* is because, in Appendix 1, that path was configured as what's known as the *Application Root Path*. The Application Root Path is the directory under which EWD's compiler will search for EWD applications. The name of each application is the same as the name you assign to each subdirectory under the Application Root Path. We named our application ***stdemo***, so that's why we created the files in the directory *c:\ewdapps\stdemo*.

Now let's take a close look at the contents of *index.ewd*. You'll notice that the EWD files we created just consist of XML (or XML-like) tags. In fact, as we'll see later, an EWD file can also contain HTML tags and Javascript. Those of you who are familiar with technologies such as PHP or Java Server Pages will therefore realize that EWD is essentially what's generally known as a "server pages" technology. However, since most of the EWD pages are just fragments of markup that are injected into the Container Page, we refer to EWD as a "server fragments" technology. EWD differs significantly from other server pages technologies in many ways. In particular, the level of abstraction – the degree to which EWD describes <u>what</u> your application should do rather that <u>how</u> it should do it – is very much higher than anything else in the web application development framework marketplace. That means your type less and your EWD files are a very succinct description of what each fragment is going to be doing.

Here's *index.ewd* again:

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

<st:container rootPath="/sencha-1.0/" contentPage="helloworld"
title="Hello World">
  <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/kit-
chensink/resources/img/phone_startup.png" />
    <st:image type="icon" src="/sencha-1.0/examples/kitchensink/re-
sources/img/icon.png" addGloss="true" />
  </st:images>
</st:container>
```

The tags in this file are known as Custom Tags.

The first one (<ewd:config>) is a generic EWD one which all EWD pages must start with. The attribute *isFirstPage="true"* tells EWD that this is the first page and can therefore be invoked using a simple "static" URL. The attribute *cachePage="false"* tells EWD to add a variety of HTTP response headers that will prevent your browser from caching the page, so you get a fresh copy loaded every time you invoke it.

The second Custom Tag (<st:container>) and its child tags will generate all the HTML for the Container Page. You simply need to specify a few attributes for your specific application:

- **title**: the HTML title and also the text to be used by default in the toolbar that you'll see at the top of the page in your iPhone. It's also used as the default text for the touch (startup) icon.

- **contentPage**: the name of the EWD page (fragment) that will be automatically loaded into the empty container page. Note that you don't add the .ewd file extension.

- **rootPath**: the name of the subdirectory under the web server's root directory where you installed the Sencha Touch files. This is used by EWD to generate the URLs that will fetch the Sencha Touch Javascript and CSS files.

  Inside the <st:container> tags are a set of <st:image> tags that define the URLs for the startup images and icon:

- **touchIcon**: the path of the icon image file that a user can add to their iPhone's Home Screen. This image file should be 72 X 72 pixels in size and without rounded corners or gloss (EWD can add these automatically). You'll find the

example file we're referencing is included in the Sencha Touch "Kitchen Sink" example source directory.

- **phoneStartupScreen**: the path of the startup splash screen that will appear in a phone while the first page is loading. This image file should be 320 X 460 pixels in size. You'll find the example file we're referencing is included in the Sencha Touch "Kitchen Sink" example source directory.

- **tabletStartupScreen**: the path of the startup splash screen that will appear in a tablet or desktop browser while the first page is loading. This image file should be 768 X 1004 pixels in size. You'll find the example file we're referencing is included in the Sencha Touch "Kitchen Sink" example source directory.

-

Now let's look in more detail at *helloWorld.ewd*. Here's what it contained:

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

<st:panel fullscreen="true" html="Hello World!" />
```

The first thing to notice is the <ewd:config> tag. Unlike the Container page, the attribute *isFirstPage="false"* specifies that this is <u>not</u> a first page, so can only be accessed via tokens that EWD automatically tokenizes with randomly-generated name/value pairs. What this means is that even if a user was aware that your application included a fragment named helloWorld.ewd, they would be unable to access that fragment arbitrarily via its associated URL – it can only be accessed if the URL includes the name/ value pairs that EWD is expecting to also be attached to the URL. EWD applications are highly secure as a result of this feature.

Notice also the *pageType="ajax"* attribute. *helloWorld.ewd* is known as a **fragment** because it does not contain a complete page of HTML: it is simply a fragment of content that will be injected, using Ajax techniques, into the main container page. You don't need to worry about how that happens: EWD will look after it for you. All your EWD fragments should use the same <ewd:config> tag as we've used for *helloWorld.ewd*.

The second tag in *helloworld.ewd* is an <st:panel> tag. Panels are the primary components that you use to define the UI of a Sencha Touch application. As you'll discover, there are many pre-defined types of panel, and panels can be nested inside each other to pretty much any depth. In our simple Hello World application, we are using the simplest panel possible. We've told it to occupy the full screen, which is what your main outer panel should always do, and we're just defining its content using a simple html attribute.

The result, as you saw, was a pretty unimpressive looking Hello World message! However, we can now begin to extend this application to make it look more like a proper application.

EWD : Sencha Touch Custom Tags Tutorial. Build 844: 04 Febuary 2011.

Copyright ©2011, M/Gateway Developments Ltd. All Rights Reserved

9

Lesson 3: Some Simple Changes to Hello World

In this lesson we're going to make some simple modifications to our Hello World application.

Let's first add a toolbar.

We can add a toolbar by using the <st:toolbar> tag which can be added inside any <st:panel> tag. Toolbars can be docked to a variety of places in its parent panel, but the top and bottom are the usual places. We'll dock ours to the top of the main panel.

Let's try it. Edit the fragment *helloWorld.ewd* so it now looks like:

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

<st:panel fullscreen="true" html="Hello World!">
  <st:toolbar dock="top" title="Sencha Touch" />
</st:panel>
```

Re-compile this fragment file using:

GT.M: d compilePage^%zewdAPI("Stdemo","helloWorld")
CSP: d compilePage^%zewdAPI("Stdemo","helloWorld",,"csp")
WebLink: d compilePage^%zewdAPI("Stdemo","helloWorld",,"wl")

Note those two commas again if you're using CSP or WebLink! The compilePage command just recompiled the helloWorld fragment, leaving everything else unchanged.

Now try re-running the application by clicking on the EWD touch icon again. This time you should see a toolbar at the top of the screen:

Already that looks better!

Next we'll add another toolbar, but this time we'll dock it the bottom of the main panel, and we'll also make the main content panel scrollable:

```
<st:panel fullscreen="true" html="Hello World!" scroll="vertical">
  <st:toolbar dock="top" title="Sencha Touch" />
  <st:toolbar dock="bottom" />
</st:panel>
```

Recompile and re-run by clicking the EWD icon and you should see:

Try swiping inside the main panel – you should find the text will scroll up and down and bounce back into position.

EWD : Sencha Touch Custom Tags Tutorial. Build 844: 04 Febuary 2011.

Copyright ©2011, M/Gateway Developments Ltd. All Rights Reserved

12

Lesson 4: Buttons, Events, Hiding and Showing UI Elements

We're now going to take a look at how you can add buttons into our example page's top toolbar and how they can be used to control the UI.

Edit *helloworld.ewd* so it now looks like this:

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

<st:panel fullscreen="true" html="Hello World!" scroll="vertical">
  <st:toolbar dock="top" title="Sencha Touch">
    <st:toolBarButton text="Hide" id="hideBtn" />
  </st:toolbar>
  <st:toolbar id="bottomToolbar" dock="bottom" />
</st:panel>
```

So we've added a toolbar button tag inside the top toolbar. Notice that we've also added an id attribute to the bottom toolbar: you'll see why later.

Recompile this page and try running the application again.

You should now see a button with the word "Hide" in it. Clicking or tapping it won't do anything however: that's because we haven't defined a handler for it. So let's add that now. Initially we'll just show an alert message when the button is tapped:

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

<st:js at="top">
 EWD.sencha.onHideBtnTapped = function() {
   Ext.Msg.alert('Attention!', 'You clicked the Hide button!', Ex-
t.emptyFn);
 };
</st:js>

<st:panel fullscreen="true" html="Hello World!" scroll="vertical">
  <st:toolbar dock="top" title="Sencha Touch">
    <st:toolBarButton text="Hide" id="hideBtn" handler="EWD.sencha.on-
HideBtnTapped" />
  </st:toolbar>
  <st:toolbar id="bottomToolbar" dock="bottom" />
</st:panel>
```

OK so what does this extra stuff do? We added a *handler* attribute to the "Hide" toolbar button and named it *EWD.sencha.onHideBtnTapped*. We then defined this as a function in the <st:js> tag. The <st:js> tag allows us to add Javascript to the fragment's payload. The at="top" attribute ensures that it is sent before any of the Sencha Touch Javascript

that is generated from the <st:panel> etc tags. This is important in this instance because the handler function must be defined and exist before it is attempted to be bound as a handler to the button.

This also raises an important point about the Sencha Touch tags. Unlike many of the more conventional EWD Custom Tags which generate other HTML tags, the Sencha Touch tags mainly generate Javascript. Therefore if you also add your own Javascript into a fragment, you need to determine whether it should be generated before or after the Sencha Touch code. Use the at attribute in the <st:js> tag to control this:

> <st:js at=”top”> generates your Javascript before the Sencha Touch code
> <st:js at=”bottom”> generates your Javascript after the Sencha Touch code

So, compile and run this new version. Now when you click the “Hide” button you’ll see a Sencha Touch alert:

In fact there is no reason why the handler function needs to be in the dynamically-generated EWD fragment. It would be better if it was defined in a static Javascript file for a variety of reasons, not least because the browser can cache it, but also because it will be easier to maintain and debug. So let’s do that.

Create a file named stdemo.js in your web server’s root directory (eg c:\inetpub\wwwroot or /var/www), containing the handler function:

EWD : Sencha Touch Custom Tags Tutorial. Build 844: 04 Febuary 2011.

Copyright ©2011, M/Gateway Developments Ltd. All Rights Reserved

14

```
EWD.sencha.onHideBtnTapped = function() {
   Ext.Msg.alert('Attention!', 'You clicked the Hide button!', Ex-
t.emptyFn);
 };
```

and remove it from helloworld.ewd:

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

<st:panel fullscreen="true" html="Hello World!" scroll="vertical">
  <st:toolbar dock="top" title="Sencha Touch">
    <st:toolBarButton text="Hide" id="hideBtn" handler="EWD.sencha.on-
HideBtnTapped" />
  </st:toolbar>
  <st:toolbar id="bottomToolbar" dock="bottom" />
</st:panel>
```

But now we'll have to tell EWD to load the stdemo.js file. We do this in the index.ewd
page by adding a <script> tag inside the <st:container>:

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

<st:container rootPath="/sencha-1.0/" contentPage="helloworld"
title="Hello World">
  <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/kit-
chensink/resources/img/phone_startup.png" />
    <st:image type="icon" src="/sencha-1.0/examples/kitchensink/re-
sources/img/icon.png" addGloss="true" />
  </st:images>
</st:container>
```

Recompile and re-run the application. It should run identically to before.

We now know that our handler function is being correctly triggered when the "Hide" button is clicked, but now let's make it actually hide something, specifically the bottom toolbar.

Now if you remember, we added an *id* attribute to the bottom toolbar, and now you'll discover why we did that. It provides Sencha Touch with a handle that we can use to invoke its various methods. In this example we'll invoke its hide() method. The key to this is the Sencha Touch function Ext.getCmp(id) which returns the widget object with the specified id. So, change the onHideBtnTapped function in stdemo.js as follows:

```
EWD.sencha.onHideBtnTapped = function() {
   Ext.getCmp("bottomToolbar").hide();
 };
```

Because we simply changed the static Javascript file, there's no need to recompile any EWD pages this time. Just restart the application in the mobile browser. This time, when you click the button, the bottom toolbar disappears.

So how can we get it to reappear? What would be nice would be to change the "Hide" button to say "Show", and make the toolbar reappear when it's clicked, meanwhile changing the button to say "Hide" again.

It's possible to change the button text and handler dynamically, but actually it's a lot simpler to have two buttons: "Hide" and "Show", each with their own handler, and make the buttons hide and appear at the right times. Try the following:

First amend helloworld.ewd as follows, adding the new "Show" toolbar button. Note how we initially make it hidden:

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

<st:panel fullscreen="true" html="Hello World!" scroll="vertical">
  <st:toolbar dock="top" title="Sencha Touch">
    <st:toolBarButton text="Hide" id="hideBtn" handler="EWD.sencha.on-
HideBtnTapped" />
    <st:toolBarButton text="Show" id="showBtn" handler="EWD.sencha.on-
ShowBtnTapped" hidden="true" />
  </st:toolbar>
  <st:toolbar id="bottomToolbar" dock="bottom" />
</st:panel>
```

Next amend stdemo.js as follows:

```
EWD.sencha.onHideBtnTapped = function() {
   Ext.getCmp("bottomToolbar").hide();
   Ext.getCmp("hideBtn").hide();
   Ext.getCmp("showBtn").show();
};

EWD.sencha.onShowBtnTapped = function() {
   Ext.getCmp("bottomToolbar").show();
   Ext.getCmp("hideBtn").show();
   Ext.getCmp("showBtn").hide();
};
```

By making sure that the buttons also had id attributes, we can alternately turn them on and off while making the toolbar disappear and reappear.

Let's try on last trick before we finish this lesson. How can we move the buttons to the right hand side of the toolbar? That's easily done by using the <st:spacer> tag:

Simply edit helloworld.ewd as follows:

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

<st:panel fullscreen="true" html="Hello World!" scroll="vertical">
  <st:toolbar dock="top" title="Sencha Touch">
    <ewd:spacer />
    <st:toolBarButton text="Hide" id="hideBtn" handler="EWD.sencha.on-
HideBtnTapped" />
    <st:toolBarButton text="Show" id="showBtn" handler="EWD.sencha.on-
ShowBtnTapped" hidden="true" />
  </st:toolbar>
  <st:toolbar id="bottomToolbar" dock="bottom" />
</st:panel>
```

Now recompile and re-run the demo: the buttons should now be on the right side of the toolbar. Now see what happens if you put the spacer between the two buttons: now you should have the Hide button appearing on the left side of the toolbar, and the Show button will appear on the right!

Lesson 5: How the EWD Custom Tags relate to Sencha Touch Classes

In general, every EWD Sencha Touch tag is an XML representation of a corresponding Sencha Touch Class. You can find the full API documentation for Sencha Touch classes at *http://dev.sencha.com/deploy/touch/docs/*

So, for example, the <st:panel> represents the Ext.Panel class. As such, the <st:panel> can have, as an attribute, any of the simple name/value pair Config Options defined in the Sencha Touch API documentation. However, the EWD Custom Tags can often have additional EWD-specific attributes that are short-cut ways of describing commonly used behaviours, so you'll probably find yourself using very few of the available Config Options in your EWD pages.

You can supply the Config Options as attribute names. The names are not case sensitive as far as you are concerned, but EWD will ensure that they are converted into the correct valid case-sensitive names in the final compiled code.

Some of the Config Options described in the Sencha Touch API documentation are supplied as objects or arrays, eg *items, listeners*, etc. In general, such Config Options should not be used in the corresponding EWD Custom Tags since there are alternative mechanisms within the EWD Sencha Touch tags for representing them. For example, when you nest <st:panel> tags inside each other, EWD converts the inner ones into members of the *items* array of the outer ones. If you need a more complex listener that can't be represented using the built-in shortcut techniques, you can use an <st:listeners> tag and child <st:listener> tags to represent them. However this type of advanced use of the EWD Sencha Touch tags is beyond the scope of this tutorial.
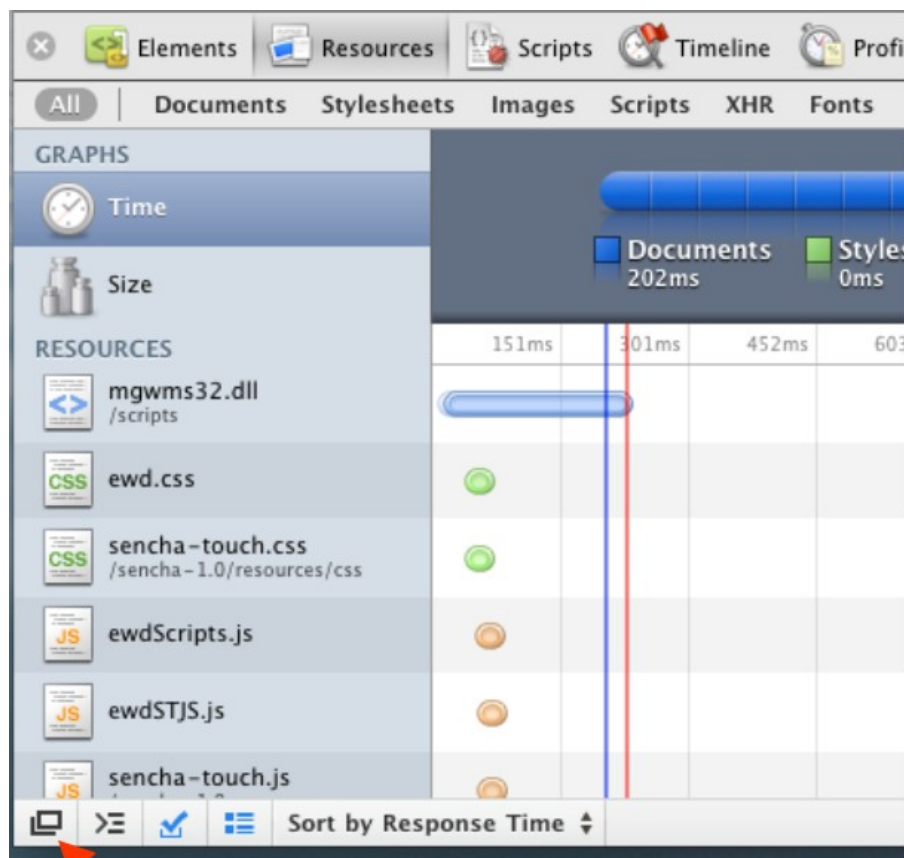
Although EWD can be used to generate Sencha Touch applications without the author really understanding much about the Javascript that EWD generates, as with any tool, the more you can learn about the Sencha Touch framework itself, the better you'll understand what's going on under the hood and the easier you'll find debugging when things inevitably go wrong.

One of the best tools you'll find for understanding and seeing what EWD is doing is the Developer Tools panel that you'll find in the desktop versions of the Chrome and Safari browsers. EWD Sencha Touch applications will run perfectly in these browsers, and you'll find that they become the preferred development platforms for your work.

To bring up the Developer Tools Panel in Chrome, click the spanner (or wrench) icon in the top left corner of the browser, and select Tools/Developer Tools from the menu:
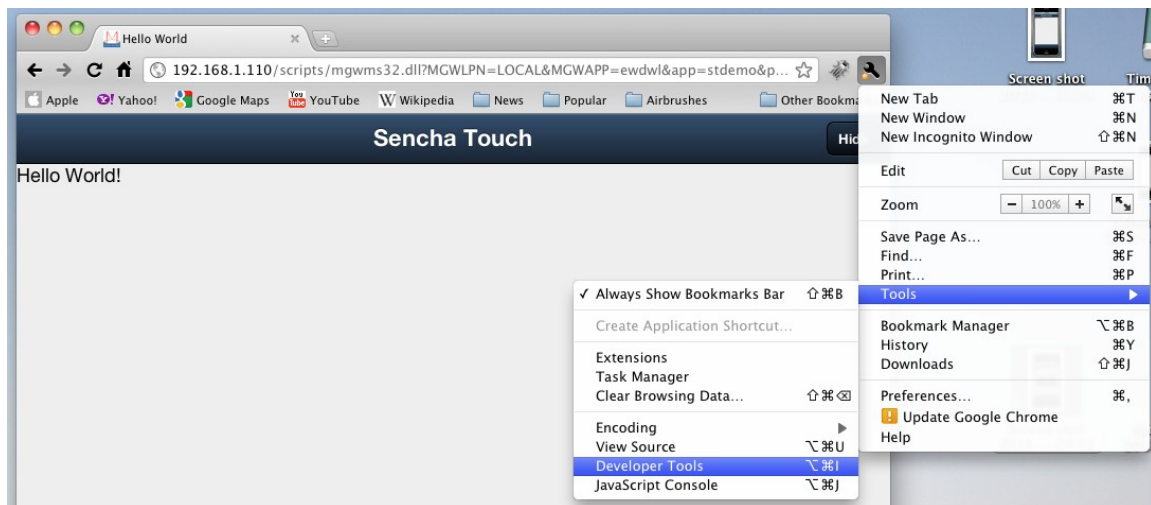
Initially you'll find the Developer Tools panel appears at the bottom half of the browser window. You'll find it's a lot easier to use if you undock it by clicking its bottom left icon:


Click this icon to undock

EWD : Sencha Touch Custom Tags Tutorial. Build 844: 04 Febuary 2011.

Copyright ©2011, M/Gateway Developments Ltd. All Rights Reserved

19

You'll also find it best to open up the Console sub-panel by clicking the icon next to the undock icon. If any Javascript errors occur you'll see lots of useful detail in this Console panel – this is invaluable when trying to debug problems:
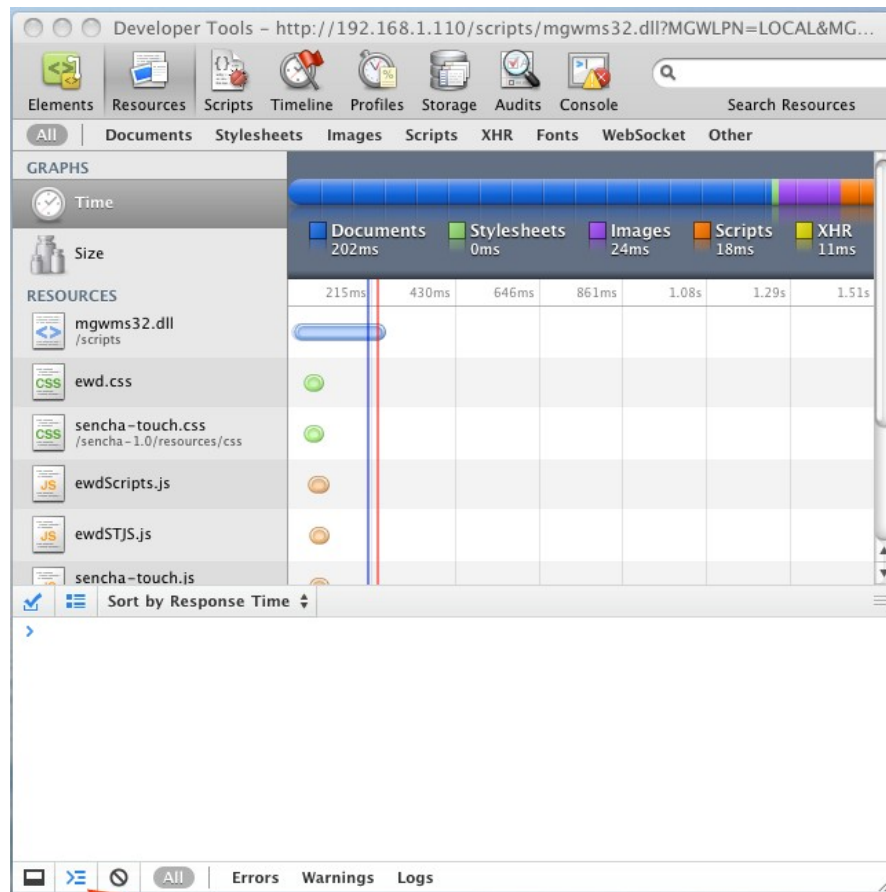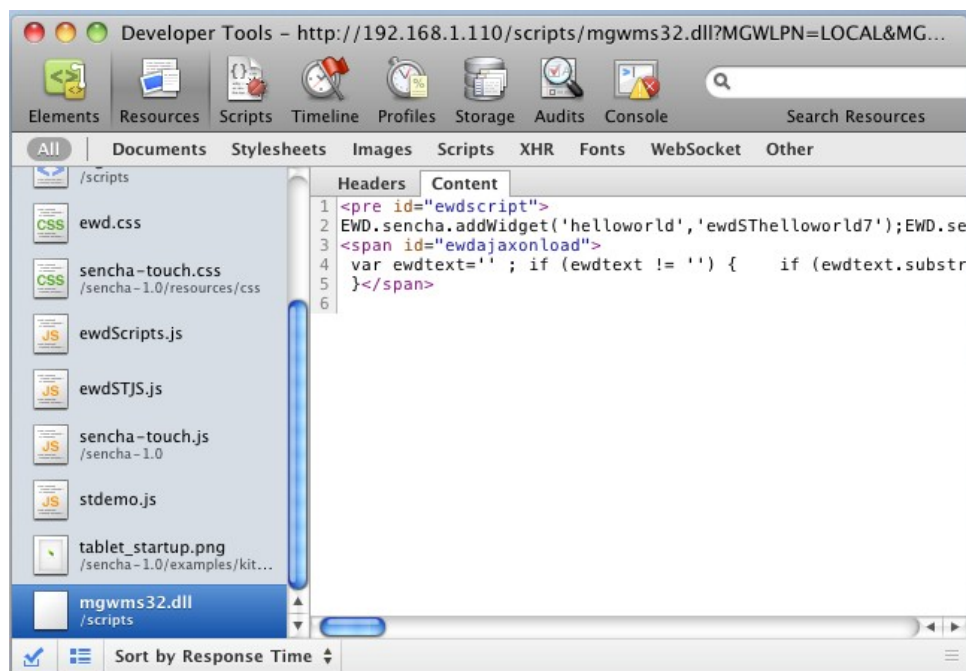

Click this icon to open the console panel

If you click on the Resources tab (as shown above) you can see the source code that was sent to the browser for each page or file that was fetched. As I'm using WebLink in the example above, my EWD pages are all named mgwms32.dll in the resource list. Click on the first one and you'll see the contents of the Container page (index.ewd) that was actually sent to the browser:
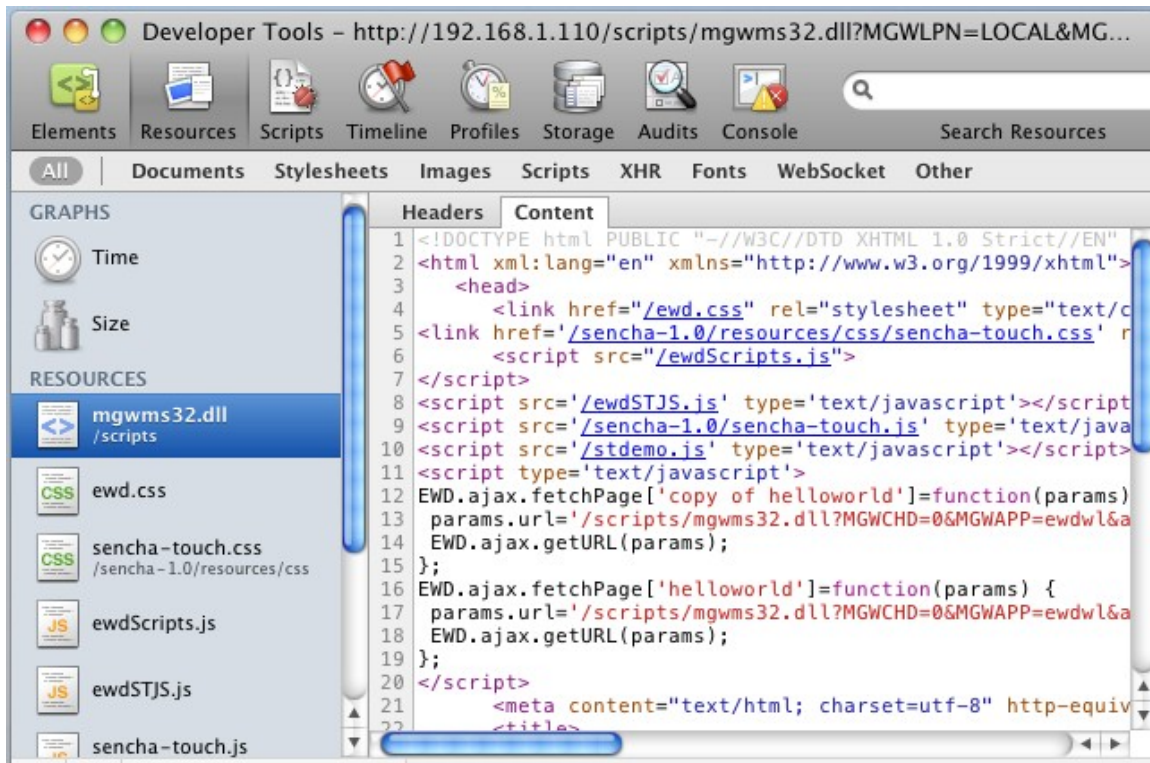
You can cut and paste the content into a text editor if you want to view it in detail.

Now find the second EWD page in the resources list, the one that represents our helloworld.ewd page, and click on it:

EWD : Sencha Touch Custom Tags Tutorial. Build 844: 04 Febuary 2011.

Copyright ©2011, M/Gateway Developments Ltd. All Rights Reserved

21

You'll see a set of long lines of Javascript inside a <pre id="ewdscript"> tag. It's pretty difficult to read and understand, so here's a tip. Copy the Javascript block (just one line commencing "EWD.sencha.addWidget" in the example above), bring up the excellent online Javascript beautifier page (http://jsbeautifier.org/), paste the generated code into the window and click the *Beautify* button. You should now see the generated Javascript nicely laid out:

Javascript unpacker and beautifier

```
EWD.sencha.addWidget('helloworld', 'ewdSThelloworld7');
EWD.sencha.widget.ewdSThelloworld7 = new Ext.Panel({
    fullscreen: true,
    html: "Hello World!",
    scroll: "vertical",
    dockedItems: [{
        dock: "top",
        title: "Sencha Touch",
        xtype: "toolbar",
        items: [{
            xtype: "spacer"
        },
        {
            handler: EWD.sencha.onHideBtnTapped,
            id: "hideBtn",
            text: "Hide",
            xtype: "button"
        },
        {
            handler: EWD.sencha.onShowBtnTapped,
            hidden: true,
            id: "showBtn",
            text: "Show"
```

Beautify

Now you can see that what was actually sent to the browser was an Ext.Panel constructor that defined our simple Hello World panel, toolbars and buttons.

If you study the code that EWD generated, you'll begin to understand how the EWD Sencha Touch tags relate to the Sencha Touch classes and their Config Options.

In the subsequent examples of this tutorial, you should use these tools to examine more closely what's happening.

Lesson 6: Accessing Data held in GT.M or Caché

So far our demo application hasn't really done anything that couldn't have been done with static files. The whole point of EWD is to allow a seamless integration with the GT.M and Caché database environments. With EWD you can integrate with either M or Caché ObjectScript code and can access data held in either Global storage or in Caché Objects. If you are using Caché you can also use Caché SQL to query your databases. Additionally EWD will interoperate perfectly with Ensemble.

In Part 2 of this tutorial you'll start to see how powerful and slick this integration really is, but let's finish off this Part 1 of the tutorial with some simple examples of how we can access and use the GT.M or Caché database.

Let's start by replacing that *Hello World* message with some information from the GT.M or Caché database. What we'll use is the current version of EWD which you can obtain by running the command within the GT.M or Caché environment:

```
write $$version^%zewdAPI()
```

Edit the helloworld.ewd page as follows:

```
<ewd:config isFirstPage="false" pageType="ajax" prePageScript="getVer-
sion^stdemo">

<st:panel fullscreen="true" html="#version" scroll="vertical">
  <st:toolbar dock="top" title="Hello World">
    <ewd:spacer />
    <st:toolBarButton text="Hide" id="hideBtn" handler="EWD.sencha.on-
HideBtnTapped" />
    <st:toolBarButton text="Show" id="showBtn" handler="EWD.sencha.on-
ShowBtnTapped" hidden="true" />
  </st:toolbar>
  <st:toolbar id="bottomToolbar" dock="bottom" />
</st:panel>
```

What we've done is added what is known as a Pre-Page Script to the fragment. This is a GT.M or Caché function that will be invoked *before* the fragment is rendered and sent to

the browser. In EWD/Sencha Touch applications, Pre-Page scripts are the only touch-point you have with GT.M and Caché. They are what you use to fetch and marshall data from your database, and what you also use to validate posted data and save it back to your database. Although you'll be frequently using Pre-page scripts in your applications, you'll find that EWD does so much for you already that the amount of actual M or Caché ObjectScript coding you'll have to do will be pretty small, which is good news for development speed and simplicity of downstream maintenance.

If you are using Caché, Pre-page scripts can either be M-style extrinsic functions or Caché Class Methods. The example above is using the M-style extrinsic function style which is compatible with both GT.M and Caché. If you wanted to use a Caché-style Class Method, you'd express it as something like:

```
<ewd:config isFirstPage="false" pageType="ajax"
prePageScript="##class(demo.st).getversion">
```

Note that if you use Caché Class Methods, your EWD applications will only be able to be run on Caché systems. If you use extrinsic functions and stick to standard M coding and Globals, you can freely migrate your EWD applications between GT.M and Caché.

You'll also see that we've replaced the literal text *Hello World* in the <st:panel> tag's *html* attribute with a reference to what is known as an EWD Session Variable: *#version.* We'll examine the EWD Session in greater detail in Part 2 of the Tutorial. For now just follow the example and see how it works.

So now we need to create that actual Pre-page Script in GT.M or Caché. Using an appropriate editor in GT.M or Caché Studio, create the routine stdemo (eg stdemo.m in GT.M) containing the following:

```
stdemo          ;
 ;
getVersion(sessid)
 ;
 n version
 ;
 s version=$$version^%zewdAPI()
 d setSessionValue^%zewdAPI("version",version,sessid)
 QUIT ""
 ;
```

If you want to use a Caché Class Method, it would look like the following:

EWD : Sencha Touch Custom Tags Tutorial. Build 844: 04 Febuary 2011.

Copyright ©2011, M/Gateway Developments Ltd. All Rights Reserved

24

```
ClassMethod getVersion(sessid As %String) As %String
{
 s version=$$version^%zewdAPI()
 d setSessionValue^%zewdAPI("version",version,sessid)
 QUIT ""
}
```

Notice that in both examples the calling signature is basically the same: the pre-page script **must** have a **single** parameter: *sessid.* This is the unique EWD Session Id that EWD's run-time engine will automatically pass to your function. That Session Id is what provides access to the user's specific persistent Session information. EWD provides a whole range of APIs that allow you to access and manipulate the user's persistent Session data, and you see the simplest one being used here: *setSessionValue()*.

Notice also that a Pre-Page script must **always** Quit with a return value. By convention, a return value of null (ie an empty string) means that the Pre-page script completed without any error occurring.

So what the pre-page script is doing is getting the EWD version string and putting it into an EWD Session value named *version.*

In the EWD page we can then access that Session variable by using the reference to it: *#version.* **When you want to use an EWD Session Variable as the value of an attribute in an EWD Custom Tag, just use the Session Variable name prefixed with #, eg as in our example:**

```
<st:panel fullscreen="true" html="#version" scroll="vertical">
  <st:toolbar dock="top" title="Hello World">
```

So, save and compile your GT.M/Caché function and then re-compile the helloworld.ewd page. Now run it again and you should see:

There we go: our application is now displaying data that has originated in the GT.M or Caché database!

Let's just make one final change to demonstrate a different way of using EWD Session Values, this time within some standard HTML markup in our panel. We can do that by removing the html attribute and simply nesting some HTML markup inside the panel, eg:

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

<st:panel fullscreen="true" scroll="vertical">
  <st:toolbar dock="top" title="Hello World">
   <st:spacer />
   <st:toolBarButton text="Hide" id="hideBtn"
handler="EWD.sencha.onHideBtnTapped" />
   <st:toolBarButton text="Show" id="showBtn"
handler="EWD.sencha.onShowBtnTapped" hidden="true" />
  </st:toolbar>
  <st:toolbar id="bottomToolbar" dock="bottom" />

  <div>
   <div>
    Your version of EWD is:
   </div>
   <div>
    <?= #version ?>
   </div>
  </div>

</st:panel>
```

Note that the block of markup must be encased in a single HTML tag: in this example
we're using a <div> tag. Inside that tag you can have any HTML markup you want.

This time, we're going to display the contents of the EWD Session variable named
version as some text inside a <div> tag. So this time we use the special syntax:

        <?= #version ?>

Those of you who are familiar with PHP will recognize this syntax. Note that the Session
Variable name must still be prefixed with a # to denote that this is a Session variable.

So, finally, compile and run the modified version of helloworld.ewd above and you
should see:

We've now covered most of the basics of building mobile web applications with EWD and Sencha Touch. In Part 2 we'll start to look at some of the cool panel and widget types that allow you to build very sophisticated applications that are every bit as powerful as Native applications.

APPENDIX 1

Installing and Configuring EWD

Installing EWD

GT.M

You should either use the M/DB installer which will fetch the very latest build of EWD automatically, or get the latest EWD routine files from https://github.com/robtweed/EWD. See our website (http://www.mgateway.com) for details on installing EWD on GT.M systems.

Caché

You should download a copy of the latest version of EWD from our web site (http://www.mgateway.com):

- Click the ***Enterprise Web Developer*** tab
- Click the tabs ***Download EWD*** followed by ***EWD for Caché***.
- Complete the registration form and you'll be able to download the latest copy of EWD for free. The Sencha Touch custom tags are included in EWD.

The zip file that you'll download contains one critical file:

- **zewd.xml** - the object code file that you install into your %SYS namespace using $system.OBJ.Load. Let this overwrite any existing copy of ^%zewd* routines if you already have EWD on your Caché system

Configuring EWD

EWD can generate CSP, WebLink and GT.M versions of Mobile web applications from the same EWD application source code. If you're already using EWD, then you can immediately start developing EWD applications.

If you're new to EWD, then you'll need to configure EWD for either WebLink, CSP or GT.M, depending on which technology you use. There are configuration instructions on our web site, but here's a quick way of configuring them, based on certain assumptions - just change the references according to your exact GT.M or Caché/WebLink/CSP configuration.

Caché & CSP

Assumptions:

- you'll be running your EWD-generated CSP applications in your USER namespace

- you're using IIS as your web server and its root path is c:\inetpub\wwwroot

- your source EWD applications will reside under the path c:\ewdapps

- the CSP application directories and files generated by EWD will be saved under c:\InterSystems\Caché\CSP\ewd

Create a global named ^zewd as follows (adjust as necessary):

```
^zewd("config","RootURL","csp")="/csp/ewd"
^zewd("config","applicationRootPath")="c:\ewdapps"
^zewd("config","backEndTechnology")="Cache"
^zewd("config","frontEndTechnology")="csp"
^zewd("config","sessionDatabase")="Cache"
^zewd("config","defaultFormat")="pretty"
^zewd("config","outputRootPath","csp")="c:\InterSystems\Cache\CSP\ewd"
^zewd("config","jsScriptPath","csp","mode")="fixed"
^zewd("config","jsScriptPath","csp","path")="/"
^zewd("config","jsScriptPath","csp","outputPath")="c:\Inetpub\wwwroot"
```

Next, create a CSP Application named "/csp/ewd" that points to the outputRootPath above and directs you to the required namespace (USER).

Caché & WebLink

Assumptions:

- you'll be running your EWD applications in your USER namespace

- you're using IIS as your web server and its root path is c:\inetpub\wwwroot

- your source EWD applications will reside under the path c:\ewdapps
-
- you'll be using the WebLink Server (MGWLPN) USER which, by default, connects incoming requests to the USER namespace

Create a global named ^zewd in the USER namespace as follows (adjust as necessary):

```
^zewd("config","RootURL","wl")="/scripts/mgwms32.dll"
^zewd("config","applicationRootPath")="/usr/ewdApps"
^zewd("config","backEndTechnology")="Caché"
^zewd("config","defaultFormat")="pretty"
^zewd("config","defaultTechnology")="wl"
^zewd("config","frontEndTechnology")="wl"
^zewd("config","jsScriptPath","wl")="fixed"
^zewd("config","jsScriptPath","wl","mode")="fixed"
^zewd("config","jsScriptPath","wl","outputPath")="c:\Inetpub\wwwroot"
^zewd("config","jsScriptPath","wl","path")="/"
^zewd("config","sessionDatabase")="Caché"
```

You also must create the global (again in USER):

```
^MGWAPP("ewdwl")="runPage^%zewdWLD"
```

This latter global creates the WebLink dispatcher to EWD's WebLink run-time engine.

GT.M

Assumptions:

- you'll be running your EWD applications in an instance of GT.M running in */usr/local/gtm/ewd*

- you're using Apache as your web server and its root path is */var/www*

- your source EWD applications will reside under the path */usr/ewdapps*

- *m_apache* has been installed and configured to dispatch to EWD's runtime code when URLs are encountered containing */ewd*

- Javascript and CSS files that are generated by EWD will be saved under the webserver path */resources*

Create a global named ^zewd as follows (adjust as necessary):

```
^zewd("config","RootURL","gtm")="/ewd/"
^zewd("config","applicationRootPath")="/usr/ewdapps"
^zewd("config","jsScriptPath","gtm","mode")="fixed"
^zewd("config","jsScriptPath","gtm","outputPath")="/var/www/resources"
^zewd("config","jsScriptPath","gtm","path")="/resources/"
^zewd("config","routinePath","gtm")="/usr/local/gtm/ewd/"
```

Creating EWD Pages

This tutorial will guide you through the process, but here's a quick summary of the process involved, based on the configuration settings shown above.

Having configured your EWD environment, you should now be ready to start developing. Create your new EWD application source pages in subdirectories of the Application Root Path, eg if your Application Root Path is *c:\ewdapps* and your application is named *myApp*::

- c:\ewdapps\myApp\index.ewd
- c:\ewdapps\myApp\login.ewd

You can use any text editor to create and edit these files.

To create the executable web application from these pages, you must compile them. This is most easily done using the command-line APIs that you invoke from within Caché Terminal or, if you are using GT.M, from within a Linux terminal session running the GT.M shell.

To compile an entire application (eg one named myApp):

CSP:

USER> d compileAll^%zewdAPI("myApp",,"csp")

WebLink:

USER> d compileAll^%zewdAPI("myApp",,"wl")

GT.M:

GT.M> d compileAll^%zewdAPI("myApp")

To compile one page (eg myPage.ewd) in an application (eg myApp):

CSP:

USER> d compilePage^%zewdAPI("myApp","myPage",,"csp")

WebLink:

USER> d compilePage^%zewdAPI("myApp","myPage",,"wl")

GT.M:

GT.M> d compilePage^%zewdAPI("myApp","myPage")

Running EWD Applications

You'll now have a runnable iPhone Web Application. You start it using the web server in your mobile device (eg Safari on the iPhone). The structure of the URL you'll use depends on whether you're using GT.M, WebLink or CSP:

CSP

For CSP EWD applications, the structure of the URL you'll use is:

- http://127.0.0.1/csp/ewd/[applicationName]/[pageName].csp

where applicationName is the name of your EWD application
     pageName is the name of the first page of your EWD application

for example:

- http://127.0.0.1/csp/ewd/myApp/index.csp

WebLink

For WebLink EWD applications, the structure of the URL you'll use is:

- http://127.0.0.1/scripts/mgwms32.dll?
  MGWLPN=LOCAL&MGWAPP=ewdwl&app=[applicationName]&page=[page Name]

where applicationName is the name of your EWD application
     pageName is the name of the first page of your EWD application

for example:

- http://127.0.0.1/scripts/mgwms32.dll?
  MGWLPN=LOCAL&MGWAPP=ewdwl&app=myApp&page=index

If you're using Apache, you'll typically replace */scripts/mgwms32.dll* with ***cgi-bin/nph-mgwcgi***

Of course if you're using a WebLink Server other than LOCAL, you'll also need to change the value of the MGWLPN name/value pair.

GT.M

For GT.M EWD applications, the structure of the URL you'll use is:

- http://127.0.0.1/ewd/[applicationName]/[pageName].ewd

where applicationName is the name of your EWD application
      pageName is the name of the first page of your EWD application

for example:

- http://127.0.0.1/ewd/myApp/index.ewd