Actions Tech. Note 8

Draft 1.0

Generating JavaScript

Many interesting Freeway Actions generate JavaScript. They insert code JavaScript into the HTML. This JavaScript will then provide interesting additional behaviours to the web-page. The oldest example of this is probably the Rollover, where a piece of JavaScript is added to the page to change an image when you move your mouse over a link. This tech note discusses some of the issues you will need to consider when making Actions that generate JavaScript.

JavaScript is added to the page in one of two ways. It i.e. either added to the page in the HTML between a `<script></script>` tag or it is added to an event handler. An example of this is the `onMouseOver` attribute of a link tag.

Adding JavaScript Between `<script></script>` Tags.

Generally the main body of JavaScript on a page is added using script tags. So, for example, if you wanted to add a function "foobar" to your page you would want to add some code like this:

```
<script language="javascript"><!--

function foobar()
{
  alert("Hi");
}

//-->
</script>
```

You generally would want to add this between the `<head></head>` of the page. So you could do it by just finding the `head` tag and add the text using `fwAddRaw` and friends. This would work - but it would not take account of the situation where there is already `<script></script>` on the page. So you could start by looking for a `script` tag and where it not there then add one. This would also work - but if you wanted compatibility with old browsers you would than have to find the special enclosing comment block (`<!--` `//-->`) and add your JavaScript in there. The whole thing gets quite complex very quickly.

Adding JavaScript within a `<script></script>` that is correctly commented for old browsers is something that the Actions writer will very commonly want to do. To ease the pain of this Freeway provides a method `fwAddJavaScript` to do all this. This method will find or create a `<script></script>` of the correct language, with comment block. So any time you need to add code between the `<head></head>` all you need do is find the `head` tag and call `fwAddJavaScript`.

So the code required to create our `foobar` fragment above is something like this:

```
<page-action name="foobar">
<action-javascript>

  function fwAfterEndBody()
  {
    // find the head tag
```

```
    var headTag = fwDocument.fwTags.fwFind("head");

    if (headTag )
    {
      // get a JavaScript script enclosure
      var javaScript = headTag.fwAddJavaScript();

      // generate the function
      javaScript.fwAddRawln('function foobar()');
      javaScript.fwAddRawln('{');
      javaScript.fwAddRawln('    alert("Hi");');
      javaScript.fwAddRawln('}');
    }
  }

</action-javascript>
</page-action>
```

JavaScript Versions

JavaScript is available in different versions on different browsers. Authors of JavaScript may require a certain version of JavaScript to be available. This is specified in HTML by specifying the version of JavaScript that you require. So if you wanted your JavaScript only to be available to browsers that can support JavaScript 1.2 you would specify this in the `language` attribute of the `script` tag:

```
<script language="javascript1.2"><!--

function foobar()
{
  alert("Hi");
}

//-->
</script>
```

The `fwAddJavaScript` method will handle this for you. It has an optional parameters that is the value of language tag that you require. If you do not specify a language tag you will get `"javascript"` as the default, otherwise you will get what you ask for. So if you want a `"javascript1.2"` enclosure you would ask for it like this:

```
var javaScript = headTag.fwAddJavaScript('"javscript1.2"');
```

NOTE: Notice that the call includes the quotes in the tag - you have to pass in the value of the tag which includes quotes.

More Readable JavaScript

Depending on the sort of person you are and the sort of thing you are doing you may want to indent your JavaScript. You can harness the indentation that is used by Freeway's code generation should you want to. This indentation will only appear when your Freeway preferences are set to "More Readable". This indentation is managed by the `fwIndent` and `fwOutdent` methods. You can also add line-endings in such a way that the line-endings will only appear when you to do this call `fwAddRawOpt` rather than `fwAddRawln`. The `fwAddRawln` method will only add a line ending when "More Readable" code is being generated.

Using these methods your code would look like this.

```
function fwAfterEndBody()
{
  // find the head tag
  var headTag = fwDocument.fwTags.fwFind("head");

  if (headTag )
  {
    // get a JavaScript script enclosure
    var javaScript = headTag.fwAddJavaScript();

    // generate the function
    javaScript.fwAddRawOpt('function foobar()');
    javaScript.fwAddRawOpt('{');
    javaScript.fwIndent();
    javaScript.fwAddRawOpt('alert("Hi");');
    javaScript.fwOutdent();
    javaScript.fwAddRawOpt('}');
  }
}
```

With "More Readable" turned off it would generate the following (more compact) code.

```
<script language="javascript"><!--
function foobar(){alert("Hi");}
//-->
</script>
```

NOTE: If you use `fwAddRawOpt` to generate your JavaScript be sure to test it in the browser with "More Readable" turned off before you release your action. This is very important as line breaks have syntactical properties and code that is, say missing a semicolon may work fine in "More Readable" form but generate JavaScript errors otherwise.

Using Markup

Adding anything more than a small function using `fwAddRawln` and friends is slow. You have to remember to quote things correctly, escape your quotes as well as typing a lot of things which are required but feel like garbage. The whole thing takes a lot of time. Then when you look in the browser and have to come back to your Action and correct the code it all becomes very tricky.

The alternative way of doing is to use `custom` markup in your action. You can put any markup you require in a `custom` markup thus

```
<action-markup custom name="foobar">
function foobar()
{
  alert("Hi");
}
</action-markup>
```

Custom markup is not automatically included in the output and you refer to it in your JavaScript by name e.g.

```
alert(fwMarkups["foobar");
```

NOTE: remember to include the `custom` or it will be included in the output which is almost certainly what you don't want.

Custom markup is designed to allow you to enter text, for use within an action, without a lot of pain. Using this technique your action would look like this.

```
<page-action name="foobar">

<action-markup custom name="foobar">

  function foobar()
  {
    alert("Hi");
  }

</action-markup>

<action-javascript>

  function fwAfterEndBody()
  {
    // find the head tag
    var headTag = fwDocument.fwTags.fwFind("head");

    if (headTag )
    {
      // get a JavaScript script enclosure
      var javaScript = headTag.fwAddJavaScript();

      // add the foobar code
      javaScript.fwAddRawOpt(fwMarkups["foobar"]);
    }
  }

</action-javascript>
</page-action>
```

If you have simple parameter substitutions that you need in your markup you can use the `fwSubstitute` method on your markup. For some sorts of actions you may find this a very easy way of way to put parameters in your functions. The following action customises the `foobar` method with text drawn from a parameter.

```
<item-action name="foobar">
<action-text name="greeting" default="Hi!">

<action-markup custom name="foobar">

  function foobar()
  {
    alert("&greeting;");
  }

</action-markup>

<action-javascript>

  function fwAfterEndBody()
  {
    // find the head tag
    var headTag = fwDocument.fwTags.fwFind("head");

    if (headTag )
    {
      // get a JavaScript script enclosure
```

```
        var javaScript = headTag.fwAddJavaScript();

        // add the foobar code substituting the parameters
        // first
        javaScript.fwAddRawOpt(fwSubstitute(fwMarkups["foobar"]));
      }
    }

  </action-javascript>
  </item-action>
```

Adding Common Code

It is common for more than one action to be applied to a page. These actions may well want use common functions. If you are in the rollover business you may well have written an image-swap function. You probably do not want to replicate this function for every rollover on the page - it would be better to add it just the once and then have all the interested actions use it.

The issue here is who should add the code and how should the other actions 'know' not to add the code. There are many ways that you could do this by the approach we have taken (in our actions) for this issue is that every action should try and add the code if no other action has added it already. The way that we do this is to set a property on the page with the same name as the function when have added the function. The following code fragment handles this:

```
// get a JavaScript script enclosure
var javaScript = headTag.fwAddJavaScript();

// has the foobar code NOT been added?
if (!fwPage.foobar)
{
  // add the foobar code
  javaScript.fwAddRawOpt(fwMarkups["foobar"]);

  // mark the page so no other action
  // will add it
  fwPage.foobar = true
}
```

Adding code like this based on a custom markup item is very common. If you want to do this you might consider pasting the following method into your action as it handles it all for you. It will check and add a property to the page with the same name as your markup.

```
// This appends a piece of JavaScript stored in /action-markup/ to a specific tag
function AppendJavaScript(tag, markup, script)
{
  // Append a piece of markup if it is not already defined
  if (tag && !fwPage[markup])
  {
    var javascript = (script) ? tag.fwAddJavaScript('"'+script+'"')
      : tag.fwAddJavaScript();
    javascript.fwAddRawOpt(fwMarkups[markup]);
    fwPage[markup] = true;
  }
}
```

If you choose to use this method adding JavaScript from mark-up is very easy - just call `AppendJavaScript` with the tag and the name of the mark-up and the rest is handled for you. The action's body looks like this.

```
function fwAfterEndBody()
{
  // find the head tag
  var headTag = fwDocument.fwTags.fwFind("head");
  AppendJavaScript(headTag, "foobar");
}
```

If you want to add JavaScript 1.2 you can do this as well, just specify the tag contents (without quotes to make life easier) and it will happen

```
AppendJavaScript(headTag, "foobar", "javascript1.2");
```

It may well be that you have different actions that share the same code. You can use the same technique but you will obviously have to duplicate the code that you want as markup in each of the actions. If you do this you will have to be a little careful that all the code is the same otherwise you can have unexpected problems.

Generating Code from more than one Action

You may well have actions that generate code together. You might have a situation where a number of actions "club together" to generate a single method. An example of this is a single method that, when called, will preload all the images. It may be that you want code like this generated:

```
function Foobar()
{
  FoobarPreload("Resources/graphic1.gif");
  FoobarPreload("Resources/graphic2.gif");
  FoobarPreload("Resources/graphic3.gif");
  FoobarPreload("Resources/graphic4.gif");
}
```

Where each call to `FoobarPreload` originates from a different Action.

A way that you can achieve this is to divide the task into two sections

- Acquiring details of all the things that need to be preloaded
- generate the code

Each action can be responsible for submitting it's own details of what needs to be added, then when all have been added one of them can generate the function. We can then split the task over two separate phases of the publishing process. At `fwBeforeStartBody` each action can add whatever it wants to be preloaded to a list. Then by the time we get to `fwBeforeEndBody` we can guarantee that everyone who wants to has added to this and the first action that is about can generate the code and clear the list so no other actions will try and generate the magical `foobar` function.

Here is a simple example:

```
<item-action name="foobar">
<action-file name="preload"/>

<action-javascript>

  function fwBeforeEndBody()
  {
    var preloadFile = fwParameters["preload"];
```

```
    // check we have a file
    if (!preloadFile.fwHasFile)
      return;

    // is there a list called 'foobar' on the page - if not
    // create it.
    if (!fwPage.foobar)
      fwPage.foobar = new Array;

    // add the file to the list
    fwPage.foobar.push(preloadFile);
  }

  function fwAfterEndBody()
  {
    // find the head tag
    var headTag = fwDocument.fwTags.fwFind("head");
    if (!headTag)
      return;

    // the page has a foobar list this means that nobody
    // has made the foobar function yet - so let's do it.
    if (fwPage.foobar)
    {
      // get a JavaScript script enclosure
      var javaScript = headTag.fwAddJavaScript();

      // generate the function foobar
      javaScript.fwAddRawOpt('function foobar()');
      javaScript.fwAddRawOpt('{');
      javaScript.fwIndent();

      for (var i in fwPage.foobar)
        javaScript.fwAddRawOpt('FoobarPreload(\'', fwPage.foobar[i], '\');');

      javaScript.fwOutdent();
      javaScript.fwAddRawOpt('}');

      // clear the foobar list so no other action
      // will try and generate the foobar function
      fwPage.foobar = null;
    }
  }

</action-javascript>
</item-action>
```

Adding JavaScript to Attributes

It is common to want to add JavaScript to attributes. HTML supports a number of event handlers to which it is possible to add JavaScript. For example you may well want to add a script the executes when your page is fully loaded. This script might, for example, call the code that is required to preload images for your rollovers.

So for this example what you would want is end up with the following body tag with an `onload` attribute set appropriately.

```
<body bgcolor="#ffffff" onload="foobar();">
```

If Freeway you can quite easily find the body tag and set it.

```
var bodyTag = fwDocument.fwTags.fwFind("body");
bodyTag.onload = '"foobar()"';
```

This will work fine but - if anyone else had set the `onload` attribute whenever they have there will be lost. Another (better) plan would be to look in the attribute - see what was there and append the method call to it (respecting the quotes). the situation is further complicated by the fact that someone could have added code like return true to the attribute. This is something that has to appear at the end of the attribute.

Freeway supplies the `fwAddJToTag` method that will correctly append to this attribute and will respect the code fragments `return true` and `return false`.

So the code:

```
var bodyTag = fwDocument.fwTags.fwFind("body");
bodyTag.fwAddJToTag('onload','foobar()');
```

will (if the attribute has not been set) would result in

```
onload="foobar()"
```

if you where to call `fwAddJToTag` twice

```
var bodyTag = fwDocument.fwTags.fwFind("body");
bodyTag.fwAddJToTag('onload','foobar1()');
bodyTag.fwAddJToTag('onload','foobar2()');
```

you would get

```
onload="foobar()1;foobar()2"
```

Notice that the `fwAddJToTag` function will automatically append a ";" between the strings that you pass it. It is sensitive to statements that have a return true or return false so

```
bodyTag.fwAddJToTag("'onload'", 'foobar1()');
bodyTag.fwAddJToTag("'onload'", 'return true');
bodyTag.fwAddJToTag("'onload'", 'foobar2()');
```

will result in

```
onload="'foobar1();'foobar2();return true"
```