Making The View: Part 6

Posted At : November 10, 2006 1:31 AM | Posted By : Cutter
Related Categories: Development, ColdFusion, Making the View

There is more than one way to skin a cat. In our last tutorial I showed you an old, tried-but-true method for including page headers and footers. But, every ColdFusion developer knows that there are many different ways. In this tutorial we'll go over creating the same header and footer using custom tags.

Custom Tags provided a huge level of power to ColdFusion Markup Language, because it gave us the ability to write simple and elaborate code snippets that could be placed in a centralized repository for consistent reuse. One way to think of them is like an include, but with expanded functionality. Let's get started.

We're going to create a new folder under our root directory titled *cftags*, and within this a subfolder titled *ui*. Within the ui folder, create a new template called *dspMainTemplate.cfm*, and copy the contents of your old *incHeader.cfm* into the file.

The first thing you want to do is make sure that the file is only called as a Custom Tag. To do this, just verify that the template has an *executionMode*, as only Custom Tags do. If it isn't called as a Custom Tag, then you'll want to stop all processing and tell someone they've made a mistake.

dspMainTemplate.cfm

```
<cfif NOT IsDefined("thisTag.executionMode")>
 <cfoutput>
  Must be called as customtag.
 </cfoutput>
 <cfabort />
</cfif>
```

Next you'll want to make sure that a closing tag was included in the calling template. If one isn't present then you'll want to throw an appropriate error message.

dspMainTemplate.cfm

```
<cfif NOT thisTag.hasEndTag>
 <cfthrow message="Missing end tag." />
</cfif>
```

You may remember that our header had *cfparam*s for the many different dynamic variables we had defined as necessary for our header and footer. Custom Tags use the **attributes** scope for any variables that are to be passed in.

dspMainTemplate.cfm

```
<cfparam name="attributes.pageTitle" default="My Site" />
<cfparam name="attributes.pageKeywords" default="Page specific keywords for SEO" />
<cfparam name="attributes.pageDescription" default="Page specific description for SEO" />
<cfparam name="attributes.pageMetaTags" default="" />
<cfparam name="attributes.additionalStylesheets" default="" />
<cfparam name="attributes.additionalScripts" default="" />
<!--- Navigation Menu variables are paramed to prevent errors --->
<cfparam name="attributes.mainMenu" default="" />
<cfparam name="attributes.sideNav" default="" />
<!--- Used by navigation menus --->
<cfparam name="attributes.siteSection" default="" />
```

Now, down to the nitty gritty. You know from above that we are making a Custom Tag that will require an ending tag. This is because your primary page content will go between the opening and closing tags, so you're going to test for the 'start' of your tag

call. That's what *executionMode* is for.

dspMainTemplate.cfm

```
<cfif thisTag.executionMode IS "start">
```

Next you'll have your actual header content, replacing the calls to your dynamic variables, so that they will now reference the *attributes* scope, instead of the *variables* scope.

dspMainTemplate.cfm

```
<cfoutput>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
  <!--- Dynamic Page Title --->
  <title>#attributes.pageTitle#</title>
  <!--- Dynamic MetaTags --->
  <meta name="description" content="#attributes.pageDescription#" />
  <meta name="keywords" content="#attributes.pageKeywords#" />
  <cfif len(trim(attributes.pageMetaTags))>
   #attributes.pageMetaTags#
  </cfif>
  <!--- Default, site-wide style sheet --->
  <link href="css/default.css" rel="stylesheet" type="text/css" media="screen" lang="en" title="Default Site Stylesheet" />
  <!---
  // Loop through additional stylesheets (passed as a list).
  // Note the lack of 'media', 'lang', or 'title' attributes.
  // To include those attributes will cause Firefox to not load the stylesheet
  //  (as of 1.5, I haven't tested this lack of functionality in 2.0 yet)
   --->
  <cfif len(trim(attributes.additionalStylesheets))>
   <cfloop list="#attributes.additionalStylesheets#" index="variables.stylesheetName">
    <link href="css/#variables.stylesheetName#" rel="stylesheet" type="text/css" />
   </cfloop>
  </cfif>
  <!--- Global functions javascript file --->
  <script src="scripts/glFunctions.js" language="javascript" type="text/javascript"></script>
  <!--- Loop through a list of file names to include additional javascript files --->
  <cfif len(trim(attributes.additionalScripts))>
   <cfloop list="#attributes.additionalScripts#" index="variables.scriptName">
    <script src="scripts/#variables.scriptName#" language="javascript" type="text/javascript"></script>
   </cfloop>
  </cfif>
 </head>
 <body>
  <!--- div containing the entire body of the document --->
  <div id="totalBody">
       <!--- div containing the entire header --->
       <div id="headerBlock">
            <div id="headerText">
                 <h2>This is my header block</h2>
            </div>
            <div id="headerMenu">
                 #attributes.mainMenu#
            </div>
       </div>
       <div id="mainContent">
     <div id="navMenu">
     #attributes.sideNav#
    </div>
</cfoutput>
```

Wow! Can't get much easier than this. Next we need to check for the 'end' of the *executionMode* and apply our footer code.

dspMainTemplate.cfm

```
<cfelseif thisTag.executionMode IS "end">
 <cfoutput>
    </div>
```

```
   </div>

  </body>
 </html>
 </cfoutput>
</cfif>
```

It's that simple. But wait! There's more! Lucky contestants will also learn how to call the custom template! First we need to import the Custom Tag for use. By using *cfimport* you are able to import all of the tags within a specific directory as well as set a unique identifier for referring to those tags.

index.cfm

```
<cfimport prefix="ui" taglib="/blogproject/cftags/ui">
```

One of the first things you'll notice in the following code is that I am still writing many of my dynamic variables to the *variables* scope initially. This is for convenience, because many of these variables could be quite large, so it is easier to 'set' them first and then pass them to the Custom Tag's attributes.

index.cfm

```
<cfscript>
 variables.pageTitle = "My Site - Home Page";
 variables.pageKeywords = "My Site, Home Page, coldfusion, programming, XHTML, javascript, css";
 variables.pageDescription = "The My Site Home Page is the gateway into My Site, where many interesting things are bound to be discovered.";
</cfscript>
<cfsavecontent variable="variables.pageMetaTags">
 <cfoutput>
  <meta name="robots" content="index,follow" />
  <meta name="revised" content="Cutter, 11/9/06" />
 </cfoutput>
</cfsavecontent>
<cfscript>
 variables.additionalStylesheets = "home.css";
 variables.additionalScripts = "myajax.js,calcfunct.js";
 variables.siteSection = "Home";
</cfscript>
```

Next we'll create our 'start' tag, passing in all of the attributes needed by the tag. We'll follow that with some content, and then write our closing tag.

index.cfm

```
<ui:dspMainTemplate pageTitle="#variables.pageTitle#"
  pageKeywords="#variables.pageKeywords#"
  pageDescription="#variables.pageDescription#"
  pageMetaTags="#variables.pageMetaTags#"
  additionalStylesheets="#variables.additionalStylesheets#"
  additionalScripts="#variables.additionalScripts#"
  siteSection="#variables.siteSection#">

<!--- Here we will now include page content --->

</ui:dspMainTemplate>
```

Voila! It's that easy. Sure, you can make it as complicated as you wish, but we just want to cover some basics in these tutorials. Using the same template file you could also use the the old >cf_myTag< syntax as well, without importing all of the tags in a folder, but I tend to organize my tags in such a way that all of the ones I might need for certain tasks are located in one folder and only called when they are needed.

Anyway, that's it for this go around. Next time out maybe we'll start exploring our base

template from within a framework? What do ya'll think? Any suggestions?