Event Handlers Guide.

Introduction

This is a quick guide to event handlers for the ColdBox Coldfusion Framework. It will give you a quick overview of event handler syntax, regulations, locations, method invocations, and declarations. It will also show you some event handler code samples.

Event Handler Location

All event handlers should be placed in the **handlers** directory of your application.

```
|ApplicationRoot
|----+ handlers (Event  Handlers Directory
|----+        system (Coldbox System  Directory)
```

Event Handler Syntax Regulations

{event Handler CFC}.{method}
Complete Regular Expression: ^eh[a-zA-Z]+\.(dsp|do|on)[a-zA-Z]+

This looks very similar to a java method call, example: String.getLength(), but without the parenthesis. In order for the framework to execute events, you need to tell it which events to run. You can do this by setting the **"event"** variable in the request collection either through URL or FORM variables. Once the event variable is set, the framework runs a regular expression match on the event string in order to validate it. If it fails, it will then throw a framework error and stop all execution. If the match is successful, the framework will then tokenize the event string to retrieve the cfc and method call and validate it against the internal registered events. It then continues to instantiate the event handler and call the event handler's method that are registered in the interal ColdBox structures.

In order to also provide a coding methodology, event handlers will be named in the following manner:

eh{Name}.cfc
example: ehGeneral.cfc, ehUsers.cfc

Regular Expression: ^eh[a-zA-Z]+\.

It includes the prefix "eh" in order to distinguish them in your code and error messages. If you do not comply to this format, the framework will throw an invalid event handler error.

As for method syntax please see the section below.

Event Handler Method Regulations

As of now, there are only three types of event handler methods that the framework accepts for public execution, private methods can be named as you like, like it should be:

1. onMethods
2. doMethods
3. dspMethods

The framework does a regular expression match on the method name in order to validate it according to prefix.

Regular Expression: (on|do|dsp)[a-zA-Z]+

---

onMethods:

These methods are to facade the internal Coldfusion Application.cfc methods or it could be whatever you want. You declare them in your config.xml file like so:

```
<Setting  name="RequestStartHandler"
          value="ehGeneral.onRequestStart"/>
<Setting  name="RequestEndHandler"
          value="ehGeneral.onRequestEnd"/>
```

Below you will find an example onRequestStart handler:

```
<cffunction name="onRequestStart"  access="public">
  <cfif  isDefined("session.authorized")>
    <cfif not session.authorized>
      <cfset setView("vwLogin")>
    </cfif>
  <cfelse>
     <cfset setView("vwLogin")>
  </cfif>
</cffunction>
```

---

doMethods:

These are most likely your model invocation calls, depending on your application. The main purpose is for them to carry out a certain processing operation. Example: doLogin, doCreateAccount.

However, with these methods you have two alternatives of program flow described below:

1. Do your processing and continue your flow by rendering a view, using **setView()**.
2. Do your processing and set a next event (relocation). This will make the controller relocate your browser to the next event. This flow will allow you for example to make the user relocate to another section of your site, without incurring with POST submits on refresh.

1. If you are going to use the first alternative then you will need to set a view for display. This is accomplished by calling the **setView**() method of the controller. Below is an example of such event handler:

```
<cffunction name="doHello"  access="public">
  <!--- Do Your Logic  Here --->
  <cfif getValue("firstname")  eq "">
    <cfset setValue("firstname","Not  Found")>
  <cfelse>
   <cfset setValue("firstname",  getValue("firstname"))>
  </cfif>
  <!--- Set the View  To Display --->
  <cfset setView("vwHelloRich")>
</cffunction>
```

If you do not set a view in your event handler then the framework will have no view to render

and throw a view not set error.

2. As for the second program flow you will need to set a next event to be run by forcing the controller to relocate. You will do this by using the **setNextEvent()** method of the controller or you can just simple use a cflocation with an event url parameter. Below is an example of such event handler:

```
<cffunction name="doStartOver"  access="public">
    <!--- Do Your Logic  Here --->
    <cfset rtn=model.insert(getValue("fname"),getValue("lname"))>
    <!--- Relocate with  new Event --->
    <cfset setNextEvent("ehGeneral.dspUserListings")>
</cffunction>
```

---

dspMethods:

These methods are usually used to prepare a view for display. Let's say that your are preparing a view that needs two queries in order to be displayed. You will place those query model calls here and then render the view. Example: dspLogin, dspUserListings. Please note that every dspMethod needs to set a view to render, it needs to call the **setView**() method in the controller. Below is an example of such event handler:

```
<cffunction name="dspLogin"  access="public">
  <cfset var user = CreateObject("component","model.users")>
  <cfset var general = CreateObject("component","model.general")>
  <!--- Do Your Logic  Here --->
  <cfset setValue("qRoles",user.getRoles())>
  <cfset setValue("qDepartments",general.getDepartments())>
  <!--- Set the view  to render --->
  <cfset setView("vwTest")>
</cffunction>
```

How to set and get values (Event Handlers and Views)

In order for any event handler to work, it needs values. Most likely url parameters, form submissions or application/session/client variables. The framework provides you with a **reqCollection** structure for all your variable needs. The framework automatically captures the FORM and URL scopes, in specific precedence, into the reqCollection for your usage. It also provides you with utility methods to interact with the reqCollection. For more in depth methods look the the **API**

1. **getValue** (name, defaultValue)
    1. name: The name of the variable to return from the reqCollection
    2. defaultValue: The default variable to return if the variable is not found in the reqCollection. Since there are no default values that can be set for complex variables, you can send the following action keywords to return an empty complex variable according to the keyword. Please note that you need to pass in the keyword in brackets. Else the regular expression match will fail and the simple value will be returned.
        1. [array]
        2. [struct]
        3. [query]
2. **setValue** (name, value)
    1. name: The name of the variable to set in the reqCollection.
    2. value: The value of the variable (simple or complex)

So if you need to retrieve a value from a form POST or URL parameters, then you will use the **getValue**() method. If you need to store values into the reqCollection in order for the views and layouts to access them, use the **setValue**() method. Please remember that you can still

continue to use both of these scopes, FORM and URL in your event handler.

So in your views and layouts, you can use the getValue() method to retrieve variables to show or queries. Just look at the sample apps for more info.
