Chapter 1: Introduction

Overview

Fusebox

Introduction

Fusebox is a development methodology for web applications. Its name derives from its basic premise: a well-designed application should be similar to the fusebox in a house. In a fusebox, if one fuse is blown, the rest of the fuses continue to work. Each fuse has its own defined job, and Fuse A does its job without any help from Fuses B, C, or D. Similarly, in a Fusebox application, if one section of the application "breaks", the rest of the application should continue to work.

Fuseactions

In a Fusebox application, every action that an application must handle is referred to as a Fuseaction. For example, an e-commerce application would include the following possible Fuseactions:

- viewCart
- viewCatalog
- addItem
- checkOut

Fuses

To perform each Fuseaction, a Fusebox application uses a series of ColdFusion pages. Each page is referred to as a Fuse. To perform the Fuseaction 'viewCart' mentioned above, the user's cart must first be queried and then the results must be displayed. Therefore, the Fuses called for the Fuseaction viewCart would be:

- qryCart.cfm
- dspCart.cfm

This brings us to a discussion of the possible types of fuses.

Types of fuses

Fuses can be grouped into a finite number of types. The four most common are:

- Display fuses, prefixed with dsp, which are used to show information to the user.
- Select Query fuses, prefixed with qry, which contain SELECT queries
- Action Query fuses, prefixed with act, which contain INSERT, UPDATE, DELETE or other action queries.

- Location fuses, prefixed with url, which redirect the application to a new URL.

Back to the Fusebox analogy

Fuseactions are defined, with their Fuses, in one file, fbx_Switch.cfm. A sample fbx_Switch file is given below. At the end of each Fuseaction, the application is sent back to this file with the next Fuseaction. To continue our e-commerce example, after viewing the cart, the user might choose to look at the catalog to find more items to purchase or the user might choose to check out. The Fuseaction sent back to fbx_Switch would be viewCatalog or checkOut.

Sample fbx_Switch file

We have used many new terms, which can be confusing. Before looking at the files used in a Fusebox application, here is a short sample of code from a fbx_Switch file, showing Fuseactions and Fuses in use. After reviewing, you should have a better idea of what Fuseactions and Fuses are.

```
<cfswitch expression = "#fusebox.fuseaction#">


<cfcase value="showInputForm">
        <cfset XFA.add = "admin.add">
        <cfset XFA.edit = "admin.edit">
        <cfset XFA.delete = "admin.delete">
        <cfinclude template="dspInputForm.cfm">
</cfcase>


<cfcase value="showEmployeeList">
        <cfinclude template="qryEmployeeList.cfm>
        <cfinclude template="dspEmployeeList.cfm>
</cfcase>


<cfdefaultcase>
        <cfoutput>
                I received a fuseaction called <B><FONT
                COLOR="000066">"#fusebox.fuseaction#"</FONT></B> that circuit
                <B><FONT COLOR="000066">"#fusebox.circuit#"</FONT></B> doesn't have
                a handler for.
```

```
        </cfoutput>
    </cfdefaultcase>


    </cfswitch>
```

Explanation of sample fbx_Switch file

In the above file, the Fuseactions are "showInputForm" and "showEmployeeList". (Don't worry about the <cfset> statements in the "showInputForm" fuseaction, they will be explained later.) The <cfinclude> statements are the Fuses for the Fuseactions. The Fuseaction that is required can be passed to the application through a URL variable.

Chapter 2: The Core Fusebox Files

Overview

Introduction

This chapter is for those users who want to know the "innards" of Fusebox. It is not necessary for your understanding of how to use Fusebox. So, if you just want to learn how to build a Fusebox application, feel free to skip this chapter totally. When you have built a few Fusebox applications and get curious you can come back!

Files

Fusebox 3.0 is built around a library of core files. These files are:

- fbx_FuseboxXX_CFXX.cfm
- fbx_Circuits.cfm
- fbx_Settings.cfm
- fbx_Switch.cfm
- fbx_Layouts.cfm

This chapter will describe what these files do and how to use them.

The Locator - fbx_Circuits.cfm

Introduction

The fbx_Circuits file is used by the Fusebox application to keep track of where it is in the folder structure of the application. There is only one fbx_Circuits file in an application.

Purpose of this file

Files in most applications are grouped within folders underneath the main application folder. The fbx_Circuits file aliases all the subfolders and sub-subfolders for ease of reference.

Example

The following is a sample of the code in a fbx_Circuits file:

- <cfset fusebox.Circuits.Home="myApp">
- <cfset fusebox.Circuits.Admin="myApp/Administration">
- <cfset fusebox.Circuits.Rules="myApp/Administration/Rules">
- <cfset fusebox.Circuits.Front="myApp/FrontEnd">

This allows all references to specific folders to be made using aliases. To call the Fuseaction named 'myFuseaction' in the folder myApp/Administration/Rules, you would call the index.cfm file as follows:

index.cfm?fuseaction=Rules.myFuseaction

Setting Defaults - fbx_Settings.cfm

Purpose of this file

The fbx_Settings file is used by the Fusebox application to set default values. There must be one fbx_Settings file in the root folder of the application. Individual circuits can have their own fbx_Settings files if default values need to be set only for the files in that circuit, or if a child circuit needs to overwrite a default value in a parent circuit.

Controlling the Look - fbx_Layouts.cfm

Introduction

It is often desirable to have a different 'look' for each section of a Web application. Controlling the look of each circuit is the purpose of the fbx_Layouts file. There must be at least one fbx_Layouts file in an application. Any circuit that has its own layout requirements should have its own fbx_Layouts file.

Control logic

The fbx_Layouts file sets the fusebox.layoutfile variable. You may use conditional logic within the fbx_Layouts file to control which layout file is used:

```
<cfif fusebox.IsHomeCircuit>
	<cfset fusebox.layoutfile = "homeLayout.cfm">
<cfelse>
	<cfset fusebox.layoutfile = "typicalLayout.cfm">
</cfif>
```

You may also use conditional logic to control a variable used by the layout file. For example, if typicalLayout.cfm was the layout file, and it contained a variable named myColor, the fbx_Layout file might contain this logic:

```
<cfif fusebox.IsHomeCircuit>
	<cfset myColor="hotpink">
<cfelse>
	<cfset myColor="red">
</cfif>
<cfset fusebox.layoutfile = "typicalLayout.cfm">
```

Using the fusebox.layout variable

The fusebox.layoutfile variable contains the name of the file that will define the actual layout. For example if fusebox.layoutfile is set to homeLayout.cfm as in the example above, then the code in homeLayout.cfm might look like this:

```
<cfinclude template="myHomeHeader.cfm">
#fusebox.layout#
<cfinclude template="myHomeFooter.cfm">
```

It is that simple. The fusebox.layout variable holds the output from a circuit. This will be discussed in more detail later in this guide.

Default value

If no layout file is assigned, fbx_Layouts will use the DefaultLayout.cfm file, which is included with the core Fusebox files.

The Control Switch - fbx_Switch.cfm

Introduction

The fbx_Switch file is probably the simplest file of the core files. It decides what files to include on the page, based on the Fuseaction.

Control logic

A <cfswitch> statement handles the logic for the fbx_Switch file. Here is the code from a simple fbx_Switch file:

```
<cfswitch expression = "#fusebox.fuseaction#">
        <cfcase value="showResults">
                <cfinclude template="qryEmployees.cfm">
                <cfinclude template="dspEmployeeForm.cfm">
        </cfcase>

        <cfcase value="updateTable">
                <cfinclude template="actUpdateEmployeeTable.cfm">
        </cfcase>

        <cfdefaultcase>
                <cfoutput>
                I received a fuseaction called <B><FONT COLOR = "000066"> "#fusebox.fuseaction#"
                </FONT> </B> that circuit <B><FONT
                COLOR="000066">"#fusebox.circuit#"</FONT></B> doesn't have a handler for.
                </cfoutput>
        </cfdefaultcase>
</cfswitch>
```

Choosing the Server Version - index.cfm

Introduction

Throughout this guide we mention index.cfm when discussing Fuseactions. Index.cfm is the entry point for the application; all Fuseactions are sent to it. You do not need to use the index file included with the core files, but it is recommended

Purpose of the file

The index file does only one thing; it checks the version of ColdFusion Server that is running and then <cfinclude>s the correct fbx_FuseboxXX_CFXX.cfm file, which we will discuss next.

Fusebox does not strictly require use of the index.cfm file, but it is probably the safest method to use. If you choose not to use the index.cfm file included with the core files, you can rename your fbx_FuseboxXX_CFXX.cfm to index.cfm. However, using the index file as is provides another safety net down the road

The Engine - fbx_FuseboxXX_CFXX.cfm

Introduction

This is the file that does it all. So before we discuss the file, we had better discuss its name. The file exists in three forms at the current time:

- FBX_Fusebox30_CF40.cfm
- FBX_Fusebox30_CF45.cfm
- FBX_Fusebox30_CF50.cfm

The current version of Fusebox is 3.0, which explains the 30 in the filename. The 40,45, and 50 refer to the version of ColdFusion that the application will be running on, either 4.0, 4.5, or the current (as of the time of the writing of this guide) version 5.0.

Looking at fbx_FuseboxXX_CFXX.cfm

The most important thing to keep in mind when using this file is that it is the engine of the application. By using <cfinclude> statements, the fbx_FuseboxXX_CFXX.cfm file controls the entire application, based on the values you set in other files.

At a very high-level, this file does the following:

1) Sets a number of variables and initializes some structures.
2) Includes the fbx_Circuits.cfm file, which is described above. Remember, the fbx_Circuits file translates the folder structure of the application to a simple structure named Circuits. This makes referring to folders, subfolders and sub-subfolders very easy.
3) Creates a reverse lookup of the circuits just defined. For example:
    - FB_.ReverseCircuitPath[myApp/Administration/Rules]=Rules
4) Includes the fbx_Settings.cfm file, described above, from the root application folder. The fbx_Settings file is used to set default values and parameters.
5) Gets the Fuseaction and the circuit that was sent to the application. Fuseactions are always prefixed by the circuits in which they reside. For example, a fuseaction named showLogin, defined in the fbx_Switch file in the myApp/Administration folder would be referenced as Admin.showLogin. The Fuseaction is stored in the fusebox.fuseaction variable and the circuit is stored in the fusebox.circuit variable.
6) Includes the fbx_Settings.cfm files from all the subfolders in top to bottom order. This allows child fbx_Settings files to overwrite values set in the parent fbx_Settings file.

7) Executes the Fuseaction in the correct circuit's fbx_Switch file using the values stored in the fusebox.fuseaction and fusebox.circuit variables. All output is stored in the fusebox.layout variable.

8) Includes the fbx_Layouts.cfm files from all the subfolders in bottom to top order. These files call the individual layout files, which output the contents of the fusebox.layout variable.

Chapter 3: Building Your First Fusebox Application

Overview

Introduction

In the Overview of the last chapter, I told you that you did not need to read that chapter to use Fusebox. Chapter 2 is therefore not a prerequisite to this chapter. There will be a good bit of repetition in this chapter for those who did read Chapter 2, but that is not a bad thing!

Choosing the Application

Introduction

The first application that you write using the Fusebox methodology should be a simple one. It should have only one circuit and only a few pages.

mySampleApp

We will call this sample application mySampleApp. The application will accept a user's name on a form. The form will have two submit buttons: one will simply display the user's first name and the second will display the entire name in an elaborate format.

Outlining the Application

Introduction

Organizing the actions that an application must perform is a vital first step to any development methodology.

Actions

The actions that mySampleApp has to take are:

- show a form for the user to enter his or her name,
- display the user's name in simple format, and
- display the user's name in elaborate format.

Fuseactions

We now take the actions and abbreviate them according to our naming standard:

- showInputForm
- showSimpleFormat
- showElaborateFormat

These actions are referred to as Fuseactions in Fusebox. We will use them to choose what pages, or Fuses, the application should run.

Fuses

For each Fuseaction above, we need to think about the page or pages necessary to complete the action. All Fusebox pages, called Fuses in Fusebox, should be named according to the following naming convention:

- Display fuses, prefixed with dsp, which are used to show information to the user.
- Select Query fuses, prefixed with qry, which contain SELECT queries
- Action Query fuses, prefixed with act, which contain INSERT, UPDATE, DELETE or other action queries.
- Location fuses, prefixed with url, which redirect the application to a new URL.

Additionally, layout files, which are not technically fuses, have a naming convention; they are prefixed with lay. They will be discussed later in this guide.

Each Fuse should perform only one action. Since this is such a simple application, each of the Fuseactions will have only one Fuse associated with it. In more complex applications, Fuseactions will have multiple fuses.

In mySampleApp, we will have these Fuses for each Fuseaction

Fuseaction	Fuse
showInputForm	dspInputForm
showSimpleFormat	dspSimpleForm

showElaborateFormat	dspElaborateForm

Editing the Core Files

Introduction

We have defined what the mySampleApp will do. The next step is to set up the Fusebox core files.

Download files

If you do not already have them, download the core Fusebox files from this URL:

http://www.fusebox.org/index.cfm?fuseaction=specs.fusebox30

The zip file has some subfolders in it. You want the FB3 Core Files. Put the files into a directory on the server. Name the directory mySampleApp.

Editing the fbx_Switch file

The first file we will edit will be the fbx_Switch file. The <cfcase> statements should look like this for mySampleApp:

```
<cfcase value="fusebox.defaultfuseaction,showInputForm">
	<cfinclude template="dspInputForm.cfm">
</cfcase>

<cfcase value="showSimpleFormat">
	<cfinclude template="dspSimpleFormat.cfm">
</cfcase>

<cfcase value="showElaborateFormat">
	<cfinclude template="dspElaborateFormat.cfm">
</cfcase>
```

Leave the <cfdefaultcase> statement as it is.

Explaining the changes

We added each of the three Fuseactions we defined above to one of the case statements. The existing value in the first <cfcase> statement, "fusebox.defaultfuseaction", is called when the user specifies which folder the Fuseaction is in, but not which Fuseaction. The <cfdefaultcase> is called if a user specifies a Fuseaction that is not listed. The case where the user does not specify a folder and does not specify a Fuseaction is handled in the fbx_Settings file (described below).

For each Fuseaction, we have used <cfinclude> to add the Fuses to the Fuseaction. Don't worry about the actual ColdFusion pages, we will code these later.

Editing the fbx_Layouts file

The fbx_Layouts file handles any conditional logic for layouts. In mySampleApp, we have two possible layouts based on the Fuseaction. When we first enter the app, we don't want any special layout, but when we choose one of the two buttons, we want the correct layout. Edit the fbx_Layouts file to:

```
<cfif isDefined("fuseaction")>
        <cfif fuseaction EQ "mysampleApp.showSimpleFormat">
                <cfset fusebox.layoutfile="laySimple.cfm">
        <cfelse>
                <cfset fusebox.layoutfile="layElaborate.cfm">
        </cfif>
<cfelse>
         <cfset fusebox.layoutfile="">

</cfif>

<cfset fusebox.layoutDir = "">
```

Again, don't worry about the actual ColdFusion pages; we will code them later.

Editing the fbx_Circuits file

The circuits file assigns convenient "aliases" to the folder structure of an application. mySampleApp consists of only one folder, so the code will be one line long:

```
<cfset fusebox.Circuits.mySampleApp="mySampleApp">
```

Editing the fbx_Settings file

The fbx_Settings file assigns default values and global values for the application. After editing the code should read:

```
<cfparam name="attributes.fuseaction" default="mysampleapp.showInputForm">
<cfset fusebox.suppresserrors = True>
```

The first line is used if a user calls the application without specifying a circuit and a fuseaction. The second line of code suppresses Cold Fusion error messages for Fusebox error messages. It is often helpful to set this to FALSE when having trouble debugging an application.

Creating Application-Specific Pages

Introduction

We have finished editing the core Fusebox files. Now we need to create the files we referenced in the core files.

Writing the dspInputForm.cfm page

This page is just a simple input form. Since it is a display page, it should only have code to output text to the browser; display pages should have no processing in them.

Here is a sample dspInputForm page.

```
<cfoutput>
<form action="index.cfm?fuseaction=mySampleApp.dspSimpleForm"
    method="post"
    name="frmGetUserName"
    id="frmGetUserName">
        <TABLE>
            <TR>
                <TD>    Enter your name: </TD>
                <TD><input type="Text"
                    name="UserName"
                    message="Please enter your first and last name."
                    required="Yes"
                    size="25"></TD>
            </TR>
            <TR>
                <TD><input type="submit"
                    name="submit"
                    value="Show simple name"
onClick="document.frmGetUserName.action='index.cfm?fuseaction=mySampleApp.showSimpleFormat';"></TD>
                <TD><input type="submit"
                    name="submit"
                    value="Show elaborate name"
onClick="document.frmGetUserName.action='index.cfm?fuseaction=mySampleApp.showElaborateFormat';"></TD
>
            </TR>
        </TABLE>
</FORM>
</CFOUTPUT>
```

Writing the dspSimpleFormat.cfm page

The simple format page outputs just the user's first name. This output is stored in the fusebox.layout variable.

```
        <cfoutput>
            #listgetat(Username,1," ")#
```

```
</cfoutput>
```

Writing the dspElaborateFormat.cfm page

The elaborate format page outputs the user's entire name. This output is stored in the fusebox.layout variable.

```
<cfoutput>
	Good evening, #Username#
</cfoutput>
```

Writing the laySimple.cfm page

The layout for the simple format page just returns the value of the fusebox.layout variable:

```
<cfoutput>

<table>
	<tr>
		<td>
			#Fusebox.layout#
		</td>
	</tr>
</table>
<p>
<input type="button"
    value="Return to Input Form"
    onClick="document.location.href='index.cfm'">
</cfoutput>
```

Writing the layElaborate.cfm page

The layout for the elaborate format returns the value of the fusebox.layout variable with a colored border.

```
<cfoutput>
<style>
.#Fusebox.thisCircuit# {
	border : double Green;
}
</style>

<table class="#Fusebox.thisCircuit#">
	<tr>
		<td>
			<b><font size="+3">#Fusebox.layout#</font></b>
		</td>
	</tr>
</table>
<p>
<input type="button"
    value="Return to Input Form"
```

```
        onClick="document.location.href='index.cfm'">
    </cfoutput>
```

Running mySampleApp

Introduction

We have edited the core files, and created the Fuses for the sample application. It is now time to run it and see how it works.

URL

The URL to run the application is \\YourTestServerPath\mySampleApp\index.cfm.

Troubleshooting

If you have problems and need more information than the Fusebox errors, set the fusebox.suppresserrors variable to false in the fbx_settings file. This will allow the ColdFusion errors to display.

Chapter 4: Enhancing mySampleApp

Overview

Introduction

In the previous chapter, we built a simple working Fusebox application. To make the application understandable, only the basic necessities of Fusebox were used. However, the beauty of Fusebox is that when fully implemented, individual circuits can be "drag and dropped" between applications. To do this, the application needs to be more parameterized.

Exit Fuseactions

Introduction

Every Fuse in a Fusebox has another Fuseaction that it calls. The Fuse may call this new Fuseaction through a <a href> URL variable or through the action parameter of a <form> tag. Each of these actions is called an Exit Fuseaction in Fusebox and is referred to by using the shorthand term XFA.

Purpose of XFAs

In mySampleApp, which we built in the preceding chapter, the dspInputForm page can send you to one of two Fuseactions:

- mySampleApp.showSimpleFormat or
- mySampleApp.showElaborateFormat

Instead of referring to those specific Fuseactions, if we replace them with variables, we can set the XFAs on the fly. This will help if we want to use the circuit in another application and have different XFAs

Changes to mySampleApp

To begin implementing XFAs in mySampleApp, we will change two files, dspInputForm and fbx_Switch:

Here is the revised section of code from dspInputForm. The changes are bold:

```
<TR>
    <TD><input type="submit"
        name="submit"
        value="Show simple name"
onClick=document.frmGetUserName.action='index.cfm?fuseaction=#XFA.Simple#';"></TD>
    <TD><input type="submit"
        name="submit"
        value="Show elaborate name"
onClick="document.frmGetUserName.action='index.cfm?fuseaction=#XFA.Elaborate#';"></TD>
</TR>
```

Here is revised code for fbx_Switch. The added lines are bold:

```
<cfcase value="fusebox.defaultfuseaction,showInputForm">
    <cfset XFA.simple="mySampleApp.showSimpleFormat">
    <cfset XFA.elaborate="mySampleApp.showElaborateFormat">
    <cfinclude template="dspInputForm.cfm">
</cfcase>
```

This doesn't seem very useful in such a simple application, but consider the following situation. You write a circuit that verifies logins. If you wrote it using two XFAs, XFA.loginVerified and XFA,loginFailed, you could drop it in multiple applications, and handle the two situations as appropriate for every application, without ever opening the circuit. You would just set the XFAs in the fbx_Switch file!

Using the fbx_Settings File to Parameterize mySampleApp

Introduction

As discussed earlier, the fbx_Settings file is used to set parameters and default values for variables. This section discusses common variables that are set in this file.

Self

The most common parameter value for a Fusebox application might be:

```
<cfparam name="self" default="index.cfm">
```

Setting up this parameter makes it much easier to change the name of your index file. Since this is the name of the file that users see in the URL, it is not uncommon to want to change it to something more meaningful

---

Parameterizing index.cfm in mySampleApp

To make this change, you need to add the line above to the fbx_Settings file and then do a replace index.cfm with #self# where appropriate. You must make sure that there are <cfoutput> tags around any code that uses #self#

Files to change in mySampleApp

After you add the <cfparam> tag above to the fbx_Settings file, the following files will need to be changed to implement the parameterization of index.cfm

- dspInputForm has index.cfm in the form action parameter and in both buttons' onClick parameter.
- laySimple has index.cfm in the onClick event of the button.
- layElaborate has index.cfm in the onClick event of the button.

Now you can go rename your index.cfm file to home.cfm and with one change, to the fbx_Settings file, your application will work with the following URL:

```
\\YourTestServerPath\mySampleApp\home.cfm.
```

Appendix A: Visual Description of a Fusebox Application

Overview

Introduction

A certain percentage of the population are visual learners. This section describes the execution of a Fusebox application using diagrams and code output.

Sample Application Description - Grandparents

Description of the application

The application used in this example is from the www.fusebox.org site. This application consists of a root directory named Grandparent, which contains the Parent folder, which contains the Child folder, which contains the Grandchild folder. The application shows how layout files work with nested folders.

Method of this section

I will briefly describe the Circuits, Settings, Layout and Switch files from the application, which are freely available for download at www.fusebox.org. I will then show the building of the page through the fbx_Fusebox30_CF50.cfm file based on various Fuseactions. To follow this explanation, it is recommended that you download and unzip these files to your CF server.

Grandparents File Descriptions

---

fbx_Circuits file

The application has the following in its fbx_Circuits file:

```
<cfset fusebox.Circuits.Grandparent="Grandparent">
<cfset fusebox.Circuits.Parent="Grandparent/Parent">
<cfset fusebox.Circuits.Child="Grandparent/Parent/Child">
<cfset fusebox.Circuits.Grandchild="Grandparent/Parent/Child/Grandchild">
```

---

fbx_Settings files

The application has fbx_Settings files in every folder. The file in the root folder contains the following code:

```
<!--- In case no fuseaction was given, I'll set up one to use by default. --->
<cfparam name="attributes.fuseaction" default="Grandchild.sayHi">

<!--- useful constants --->
<cfparam name="request.self" default="index.cfm">
<cfparam name="self" default="index.cfm">
<cfset suppressLayout = FALSE>

<!--- should fusebox silently suppress its own error messages? default is FALSE --->
<cfset fusebox.suppresserrors = false>

<cfif fusebox.IsHomeCircuit>
	<!--- put settings here that you want to execute only when this is the application's home circuit (for example "<cfapplication>" )--->
<cfelse>
	<!--- put settings here that you want to execute only when this is not an application's home circuit --->
</cfif>
```

The fbx_Settings files in the subfolders are identical, except that their default Fuseaction is #fusebox.thisCircuit#.main, instead of Grandchild.sayHi. The variable fusebox.thisCircuit evaluates to the current circuit.

---

fbx_Layouts

The fbx_Layouts file is identical for all four folders. Each file contains the following line of code:

```
<cfset fusebox.layoutFile = "SayWho.cfm">
```

Each of the folders has a SayWho.cfm file and that is where the layout information resides. For example, in the root folder, the SayWho file contains the following code:

```
<cfoutput>
```

```
<style>
.#Fusebox.thisCircuit# {
	border : double Silver;
}
</style>

<table class="#Fusebox.thisCircuit#">
	<tr>
		<td>
			<font size="-2">Don't mind me, I'm just the
#Fusebox.thisCircuit#</font>
		</td>
	</tr>
	<tr>
		<td>
			#Fusebox.layout#
		</td>
	</tr>
</table>

</cfoutput>
```

The other SayWho.cfm files are very similar, with changes only to the Border parameter.

Remember, the fusebox.layout variable will be populated by whatever output is generated by the Fuseaction.

---

fbx_Swith.cfm

The fbx_Switch files have various Fuseactions defined, which we will see more clearly in the next section.

Running the Application with Fuseaction = Grandchild.SayHi

Introduction

The Fuseaction Grandchild.SayHi is the default Fuseaction of the Grandparent circuit. This section will build the page that is shown when the URL /YourServerDirectory/Grandparent/index.cfm or the URL /YourServerDirectory/Grandparent/index.cfm?fuseaction=grandchild.sayhi is entered in a browser.

Index.cfm

The index.cfm contains logic to point to the correct fbx_fuseboxXX_CFXX file. We are using the fbx_fusebox30_CF50 file, so that is the file I will step through.

Section five of fbx_fusebox30_CF50.cfm

Sections one through four are the fusedocs and initialization of variables. Section Five is the first <cfinclude> statement. After Section five, the code on the page looks like this

Sections 1 - 4 Not Shown

<cfset fusebox.Circuits.Grandparent="Grandparent">

<cfset fusebox.Circuits.Parent="Grandparent/Parent">

<cfset fusebox.Circuits.Child="Grandparent/Parent/Child">

<cfset fusebox.Circuits.Grandchild="Grandparent/Parent/Child/Grandchild">

Section six

Section six is a script that creates the FB_.ReverseCircuitPath structure. After running this section, the structure has the following values.

FB_.ReverseCircuitPath[Grandparent]=Grandparent

FB_.ReverseCircuitPath[Grandparent/Parent]=Parent

FB_.ReverseCircuitPath[Grandparent/Parent/Child]=Child

FB_.ReverseCircuitPath[Grandparent/Parent/Child/Grandchild]=Grandchild

As you can see this is, as the name implies, a reverse lookup for the Circuits in the application. The following variables are also set in this section

fusebox.homecircuit=Grandparent

fusebox.ishomecircuit=TRUE

Section seven

Section seven includes the fbx_Settings file from the root folder. The code included is:

<cfparam name="attributes.fuseaction" default="Grandchild.sayHi">

<cfparam name="request.self" default="index.cfm">

<cfparam name="self" default="index.cfm">

<cfset suppressLayout = FALSE>

<cfset fusebox.suppresserrors = false>

Section eight

Section eight dissects the Fuseaction. These are the variables set after this section has run:

FB_.rawFA=Grandchild.SayHi

fusebox.fuseaction=SayHi

fusebox.Circuit=Grandchild

fusebox.TargetCircuit=Grandchild

Section nine

Section nine includes all the child folders' fbx_Settings files in top to bottom order. The fbx_Settings files are identical, with one exception. The default Fuseaction for the child folders is different:

<cfparam name="attributes.fuseaction" default="#Fusebox.thisCircuit#.main">

The following variables are set in this section:

FB_.fullPath= Parent/Child/Grandchild/

FB_.corePath= Parent/Child/Grandchild/

fusebox.ishomecircuit=FALSE

fusebox.currentpath= Parent/Child/Grandchild/

Section ten

Section ten includes the fbx_Switch file from the target circuit, Grandchild. This <cfswitch> statement causes the dsp_SayHi file to be included. The output generated by this file is stored in the variable fusebox.layout, so it is not yet visible to the browser.

At this point, the output would be the following:

Hi there!

The following variables are set in this section:

FB_.fuseboxpath=Parent/Child/Grandchild/

fusebox.IsTargetCircuit=TRUE
fusebox.IsHomeCircuit=FALSE

fusebox.currentPath=Parent/Child/Grandchild/

fusebox.rootPath=../../../

Section eleven

Section eleven loops over the paths from the target circuit back up to the root circuit. The code <cfinclude>s fbx_Layout files, which each contain the following <cfset> statement:

<cfset fusebox.layoutFile = "SayWho.cfm">

This value of fusebox.layoutfile is used to <cfinclude> the actual layout files by referring to the #fusebox.layoutfile# variable. Each layout file includes the #Fusebox.layout# content, and the output from each layout file is stored back into the #Fusebox.layout# variable, which is how the layouts are nested.

In more detail: when Section eleven starts, the #Fusebox.layout# variable contains the code (from dsp_SayHi.cfm) to create the output

Hi there!

The code then loops over the path leading back up from Grandchild to the root, Grandparent. At each level, the #Fusebox.Layout# variable is set to the output from SayWho.cfm. Since SayWho.cfm contains the #Fusebox.Layout# variable as part of its code, the code is recursive.

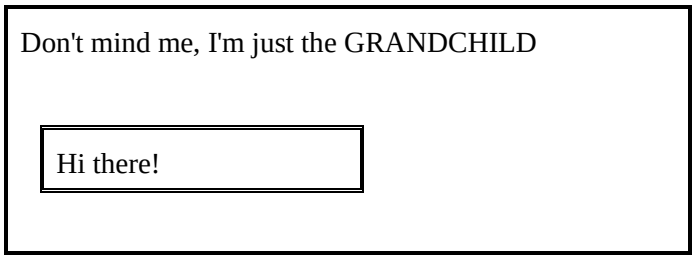
At the first loop, in the Grandchild circuit, the code <cfinclude>s this SayWho file:

```
<cfoutput>
<style>
.#Fusebox.thisCircuit# {
	border : double Silver;
}
</style>

<table class="#Fusebox.thisCircuit#">
	<tr>
		<td>
			<font size="-2">Don't mind me, I'm just the
#Fusebox.thisCircuit#</font>
		</td>
	</tr>
	<tr>
		<td>
			#Fusebox.layout#
		</td>
	</tr>
</table>

</cfoutput>
```

When this is run, #Fusebox.layout# contains the code necessary to generate:



The output of the code above is stored, in #Fusebox.layout#, will be passed into the SayWho.cfm file of the Child circuit, which will store its output back into #Fusebox.layout#, and so on.
