

Lesson 4 – the anatomy of a request

In lessons 1 through 3 we've been concentrating solely on making an application work and not considered what is actually occurring during a request, now seems like a good time to take a look at what 'magic' the core files are doing for us.

Every fusebox request begins when a request is made to the default file, usually index.cfm. In most cases all index.cfm does is cinclude the fusebox4 runtime, although it may setup `<cfapplication>`

```
<cfinclude template="fusebox40.runtime.cfm.cfm">
```

Open up the fusebox40.runtime.cfm.cfm file and follow through it.

fusebox40.runtime.cfm.cfm

Lines 32-42

The first operation to occur is that the form and url variables are copied into the attributes scope, with form variables taking precedence over url variables.

Lines 45 - 66

The internal structures fb_ and array fb_.fuseQ are defined and defaults are assigned.

Lines 69 – 80

Evaluates to false if this is the first time a request has been made to this fusebox application. If it's not the first request the mode the application is in is checked, production or development.

If the application is in development mode then load, parse and execute are all set to true, in production mode load and parse are set to false and execute is set to true.

Lines 86 – 98

If the application is in production mode, this section of code allows the developer to force a load/parse/execute at runtime by passing fusebox.password in the URL (it must match the entry in fusebox.xml.cfm) and then either fusebox.load, fusebox.parse, fusebox.execute

Lines 100 – 103

If this is the first time the application is run and application.fusebox doesn't exist this forces a load.

Lines 107 – 109

If the application wants to load or the developer has forced a load then the fusebox40.loader.cfm.cfm is included at this point.

fusebox40.loader.cfm.cfm

Lines 41-44

The application.fusebox structure is created with sub-structures, circuits, plugins and pluginphases.

Lines 55 – 59

The fusebox tests to see if fusebox.xml.cfm or fusebox.xml exists

Lines 61 – 73

fusebox.xml.cfm/fusebox.xml is attempted to be read, if it is read it is parsed to the variable fb_.application.fusebox.xml. If it can't be read or is not valid XML then an error is thrown.

Lines 77- 84

Using an xpath search the character encoding parameter is read from the config file (now a variable), if it is not found UTF-8 is assumed.

Lines 89 – 94

The config file is reread and parsed from from the disk this time using the desired character encoding.

Line 99

Time stamps the application.fusebox structure

Lines 102 – 107

The fusebox parameters are set as variables for later use.

Lines 109 – 122

Defaults for the parameters are set in case not previously set

Lines 124 – 128

Variables for the location of the parsed file output and the plugins folder are set

Lines 130 – 134

The global fuseactions, preProcess and postProcess are parsed into variables

Lines 136 – 137

Circuit definitions are retrieved from the config variable

Lines 140 – 190

Circuits are looped over adding their attributes to the internal fb_.application.fusebox.circuits structure.

For each circuit:

The filename for the circuit definition file is tested for its existence either circuit.xml.cfm (or circuit.xml) and is then read, again using the character

encoding as set in the config file. If it can't be found or is not valid XML then an error is thrown.

Lines 193 -202

A trace for each circuit is built

Lines 204 – 206

Attributes and fuseactions for each circuit are retrieved

Lines 208 – 216

The modifier for the circuit is tested. Modifiers are set in the access attribute of the circuit parameter in the circuit.xml file, if it is not assigned then it is made 'internal'

```
<circuit access="public|internal">
```

Modifiers allow developers to determine whether a circuit/fuseaction can be accessed from outside the application from a link/URL etc. In MVC style applications all requests are made via the controller and so is set public. The controller then accesses the subsequent internal circuits/fuseactions, so model and view circuits are set to internal.

Fuseactions that do not have a modifier set will inherit the modifier of their circuit

Lines 219 – 224

Permissions for the circuit are determined; this is in addition to the modifier of the circuit.

Lines 226 – 283

The circuit's fuseactions and pre/post fuseactions are determined. Fuseaction modifier and permissions are determined.

Lines 303 – 355

Plugins are determined by both both and the phase that are to operate in

Lines 360

The internal fb_.application structure is duplicated to application.fusebox

...processing returns to fusebox40.runtime.cfm.cfm

Lines 113 – 115

The plugins are looped over creating a structure named myFusebox.plugins.{plugin-name} for each one.

Lines 118 – 120

If URL variables were given precedence over FORM variables the attributes scope is appended with the url scope.

Lines 123 – 128

The value of the default fuseaction is assigned to
attributes.{fuseactionvariable}

Lines 132 – 134

Variables for myFusebox.originalCircuit and myFusebox.originalFuseaction are set based on the attributes.fuseaction variable, an error is thrown if it is malformed.

Lines 146 – 152

The circuits/fuseaction requested are tested for existence, if they don't then an error is thrown

Lines 157 – 159

Test that the fuseaction has a modifier of 'public', if it's not public then an error is thrown.

Lines 162

The file name of the parsed file to create is determined, it is of the form

parsed.{circuit}.{fuseaction}.{scriptfiledelimiter}

scriptfiledelimiter will be 'cfm' on Cold Fusion.

Lines 165 – 195

If the fusebox is set to reparse or if the parsed file doesn't exist then the transformer (fusebox40.transformer.cfm.cfm) and parser (fusebox40.parser.cfm.cfm) core files are called.

fusebox40.transformer.cfm.cfm

There's little point in delving into the transformer to a similar depth as the other core files, but briefly the API is building an array (a FuseQ) which will be passed to the parser to produce the parsed file

Lines 41 – 70

Preprocess plugins are added

Lines 74 – 95

Global preprocess fuseactions are added

Lines 97 – 106

The actual fuseaction is then added

Lines 109 – 129

Postprocess fuseactions are added, they can only be added by <do>

Lines 153 – 184

Postprocess plugins are added

Lines 185 – 215
processError plugins are added

The remainder of the transformer examines the fuseaction handling <do>'s and other special transformations such as <if> or <loop> adding them to the fuseaction.

fusebox40.parser.cfm.cfm

the parser takes the fb_.fuseQ array and loops over it parsing the fusebox XML grammar adding to the fb_.parsedfile variable 'real' cf statements, cfset, cflocation, cfincludes etc

...processing returns to fusebox40.runtime.cfm.cfm

Lines 175 – 183
If the parsed file exists then it is deleted

Lines 186 – 191
The new parsed file is written to the parsed folder from the variable fb_.parsedfile, if it cannot be written an error is thrown

Line 201
At this point the fusebox has now completed (if required) all the load and parse operations and can now enters the execution stage

Lines 202 - 216
Attempts to include the parsed file throwing an error if it doesn't exist.

The fusebox request has now completed for the requested circuit.fuseaction