

## Lesson 5 - further Fusebox XML grammar

Before we move onto looking at MVC with Fusebox4 there's a few more pieces of Fusebox XML grammar that we need to introduce.

### <set>

Set is pretty straight forward, it takes the form:

```
<set name="" value="" Overwrite(optional)
Evaluate(optional)>
```

If `overwrite` is false then it is treated as a `cfparam`, if `evaluate` is true then the `evaluate()` function is used around the value.

We can use `<set>` in our definition files to set variables for use at run time. For example if we run a query in a `fuseaction` to return the name of a page, we could do

```
<set name="request.pagetitle"
value="#myquery.pagetitle#">
```

Immediately after the `qry_` `fuseaction` before we output the results with a `dsp_`  
eg.

```
<include template="qry_myquery.cfm" />
<set name="request.pagetitle" value="#myquery.pagetitle#"
/>
<include template="dsp_mypage.cfm" />
```

We can also use `<set>` to call methods in CFCs eg,

```
<set name="myquery"
value="#somecfc.somemethod(somearguments)#" />
```

### <do>

The `do` tag is one of the most important pieces of Fusebox XML grammar when it comes to MVC applications.

It takes the form:

```
<do action="" contentvariable="" Append(optional)
Overwrite(optional)>
```

So that one's going to take some explaining. As its name suggests `<do>` is able to 'stack' multiple fuseactions into a single fuseaction.

If we have a fuseaction in a circuit called `mycircuit` that executes a `qry_` and then a `dsp_` file we could rewrite that using `<do>` as follows,

```
<fuseaction name="dodemo">
    <do action="mycircuit.query" />
    <do action="mycircuit.display" />
</fuseaction>

<fuseaction name="query">
    <include template="qry_myquery.cfm" />
</fuseaction>

<fuseaction name="display">
    <include template="dsp_mydisplay.cfm" />
</fuseaction>
```

`<do>` can also write to a content variable. So, what's a content variable? Up till now we've been dealing with fuseactions or display files that are output immediately. Content variables allow us to store the results of a `<do>` into a variable for later output, this opens up all sorts of possibilities for layouts.

Consider this example:

```
<fuseaction name="show">
    <do action="mycircuit.articles"
contentvariable="myarticles" />
    <do action="mycircuit.layout" />
</fuseaction>

<fuseaction name="articles">
    <include template="qry_getarticles.cfm" />
    <include template="dsp_articles.cfm" />
</fuseaction>

<fuseaction name="layout">
    <include template="dsp_layout.cfm" />
</fuseaction>
```

When the `show` fuseaction is requested, it calls the `articles` fuseaction but stores the results into the `myarticles` variable, then it calls the `layout` fuseaction. `Dsp_layout` would look like:

```
<html>
<title>CV demo</title>
<body>
<cfoutput>#myarticles#</cfoutput>
```

```
</body>
</html>
```

You could go on to modify the articles and layout fuseactions so their modifier is set to 'internal' so they can't be called directly as an extra precaution.

This example lets the fuseaction itself apply the layout, but we could use the post fuseaction to apply our layouts.

### **<pre/post fuseactions>**

Pre/Postfuseactions can be specified globally in the fusebox.xml.cfm file and/or on a circuit level in circuit.xml.cfm

In the fusebox.xml.cfm pre/post process fuseactions are defined in the <globalfuseactions> section, between the corresponding <preprocess> <postprocess> tags. At a global level you're limited to using <do> tags though.

In a circuit definition file you use <prefuseaction> or <postfuseaction> tags inside the <circuit> tags,

Eg:

```
<circuit name="mycircuit">

    <postfuseaction>
        <do action="mysite.layout" />
    </postfuseaction>

    <fuseaction name="...">

    </fuseaction>

</circuit>
```

Any fuseaction in the mycircuit circuit will execute but then immediately following it the postfuseaction would execute. Notice I've called a fuseaction layout which is not even in the current circuit – this is something else of importance when we come to looking at an MVC application.

### **<loop>**

Loop allows you perform a conditional loop inside your circuit definition files,

```
<loop condition="[loop condition]">
```

```

...
</loop>

```

Here's an example of a loop

```

<set name="counter" value="1"/>
<loop condition="counter LTE 3">
  <set name="bluebarwidth"
value="#int(100/counter)#" />
  <do action="layouts.bluebar" />
  <set name="counter"
value="#IncrementValue(counter)#" />
</loop>

```

This example sets the start value of counter to 1, it then calculates a bluebarwidth as a percentage, includes the layouts.bluebar fuseaction and then increments the value of counter and reiterates until the condition evaluates to true.

You're restricted to a single loop, you can't nest loops in your circuit definitions.

## <if>

If allows you to perform conditional logic in your circuit definition files, you build an <if> block looking like

```

<if condition="[expression to evaluate]">
  <>true>
    ...
  </true>

  <false>
    ...
  </false>
</if>

```

condition is the condition to evaluate, if it evaluates to true the <true> block runs, if false then the <false> block runs.

As with <loop>, <if> only lets you have a single level of conditional logic, it can't be nested.

```

<if condition="#myquery.recordcount#">
  <true>
    <include template="dsp_results.cfm" />
  </true>

```

```
<false>  
    <include template="dsp_noresults.cfm" />  
</false>  
</if>
```

This tests to see if myquery.recordcount exists, if it does it includes dsp\_results.cfm, if not it includes dsp\_noresults.cfm.