

Lesson 3 – dealing with form submissions

NOTE: This lesson assumes you've completed lesson 2, you can download the files from the previous lesson and use them as your foundation if you've skipped it.

In lesson 2 we created a two fuseaction application with a link in the first page of the application to the second fuseaction, passing a variable to the second page in the URL.

In this lesson, we're going to change the first fuseaction so it presents a form to the user allowing the user to enter their name and then greet them on the second page.

Go ahead and return to your dsp_welcome.cfm in your code editing environment.

Remove the <a href...> link and replace it with a form that asks the user for their name using a form with an input box called 'name'.

If you get stuck the code I used is below;

```
<html>
<head>
<title>Welcome</title>
</head>

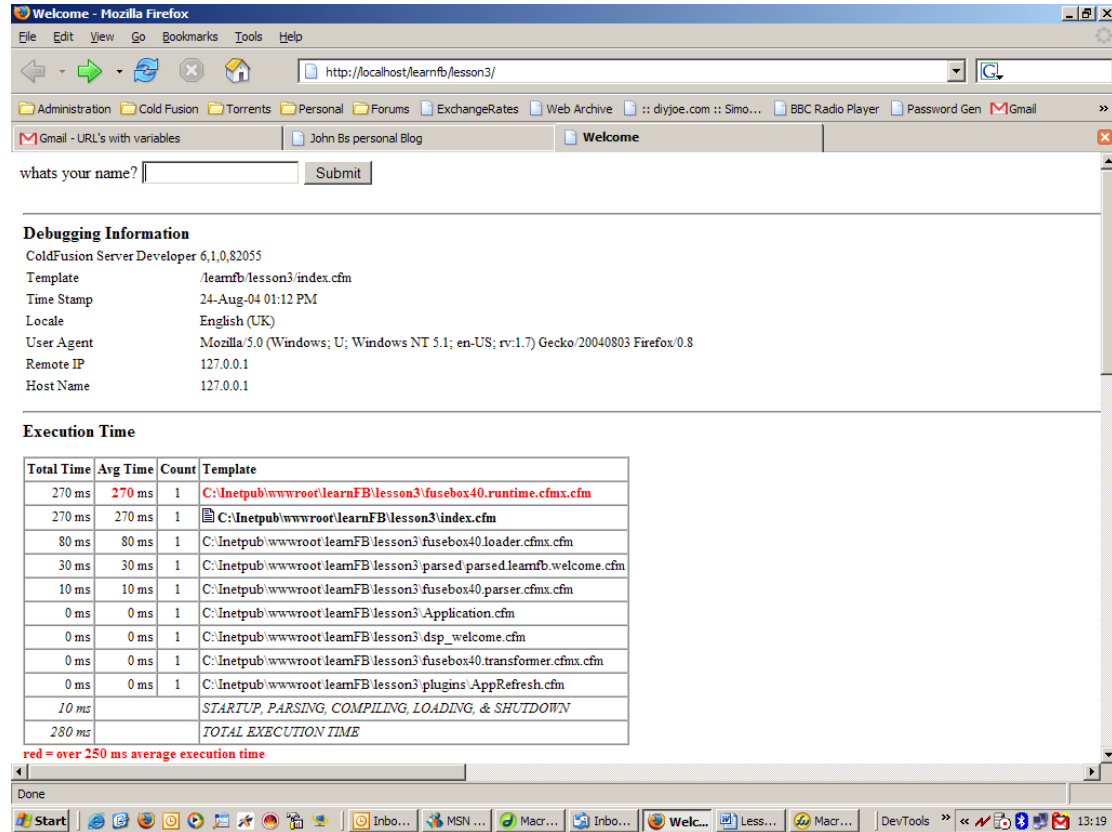
<body>
<cfoutput>
<form action="#request.self#" method="post">
<input type="hidden"
name="#application.fusebox.fuseactionVariable#"
value="#xfa.clicklink#">
What's your name?
<input type="text" name="name">
<input type="submit" value="Submit">
</form>
</cfoutput>
</body>
</html>
```

Does it look anything like yours?

Remember that every request in a fusebox application goes through the default file, usually index.cfm – just to be sure though in my code I refer to the variable we set in Application.cfm #request.self#. We also need to specify the fuseaction we want to pass the form submission to so we create a hidden form field named the same as the fuseactionVariable – again, I refer to the

value as a variable as I don't always know what will be being used. The value of this field will be the XFA value we set in our circuit definition file, #xfa.clicklink#.

If you run the application in your browser you should see the form;



Our newly created form

When you type your name into the form and click submit you see the personalised message

Hello john

Debugging Information
ColdFusion Server Developer 6,1,0,82055
Template /learnfb/lesson3/index.cfm
Time Stamp 24-Aug-04 01:20 PM
Locale English (UK)
User Agent Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7) Gecko/20040803 Firefox/0.8
Remote IP 127.0.0.1
Host Name 127.0.0.1

Execution Time

Total Time	Avg Time	Count	Template
130 ms	130 ms	1	C:\inetpub\wwwroot\learnfb\lesson3\fusebox40\runtime.cfm.cfm
130 ms	130 ms	1	C:\inetpub\wwwroot\learnfb\lesson3\index.cfm
80 ms	80 ms	1	C:\inetpub\wwwroot\learnfb\lesson3\fusebox40\loader.cfm.cfm
10 ms	10 ms	1	C:\inetpub\wwwroot\learnfb\lesson3\fusebox40\parser.cfm.cfm
10 ms	10 ms	1	C:\inetpub\wwwroot\learnfb\lesson3\parsed\parsed.learnfb.hello.cfm
0 ms	0 ms	1	C:\inetpub\wwwroot\learnfb\lesson3\Application.cfm
0 ms	0 ms	1	C:\inetpub\wwwroot\learnfb\lesson3\dsp_hello.cfm
0 ms	0 ms	1	C:\inetpub\wwwroot\learnfb\lesson3\fusebox40\transformer.cfm.cfm
0 ms	0 ms	1	C:\inetpub\wwwroot\learnfb\lesson3\plugins\AppRefresh.cfm
20 ms			STARTUP, PARSING, COMPILING, LOADING, & SHUTDOWN
150 ms			TOTAL EXECUTION TIME

red = over 250 ms average execution time

The new personalised message

But wait a second, we didn't make any changes to our receiving fuseaction 'hello' yet the application still works. That's because remember, the fusebox core copies the FORM/URL scopes into the ATTRIBUTES scope and in dsp_hello.cfm we simply referred to a variable called attributes.name. Since we named our text field 'name' in the form the value entered is available in our hello fuseaction. To confirm this, notice I used a POST method on the form, so if the scopes weren't copied to ATTRIBUTES we'd have to refer to the FORM scope on the receiving page to display the value entered by the user.

Now that we're dealing with form submissions, our XFA named clicklink doesn't really fit, so it might be an idea to change the name to SubmitForm, go ahead and change the value in circuit.xml.cfm and in dsp_welcome.cfm

You should end up with a circuit.xml.cfm looking like;

```
<circuit access="public">

    <fuseaction name="hello">
        <include template="dsp_hello.cfm" />
    </fuseaction>

    <fuseaction name="welcome">
        <xfa name="submitform" value="learnFB.hello" />
        <include template="dsp_welcome.cfm" />
    </fuseaction>
</circuit>
```

```
</circuit>
```

And dsp_welcome.cfm looking like;

```
<html>
<head>
<title>Welcome</title>
</head>

<body>
<cfoutput>
<form action="#request.self#" method="post">
<input type="hidden"
name="#application.fusebox.fuseactionVariable#"
value="#xfa.submitform#">
What's your name?
<input type="text" name="name">
<input type="submit" value="Submit">
</form>
</cfoutput>
</body>
</html>
```

So now we know how to pass data between fuseactions either via a clickable link or via a form submission we can go on to make the application a little more functional.

To do this, we need to introduce a database to our application. We'll amend our application so when a user enters a name, a db query is performed to look up the user and return the data to the user. Download the completed Lesson3 zip file and create a datasource called LearnFB pointing at the Access database. You can put the Db in the your c:\inetpub\wwwroot\learnFB folder as we'll use it in later lessons.

Assuming you've got the Db registered in the CF Administrator we can move on and query the database.

We're going to create a telephone directory search application where we can enter a name and see results matched. We'll be able to click on individual entries to view further details on the person.

I've gone ahead and made some purely cosmetic changes to dsp_welcome.cfm, just the title of the page and the text next to the text box;

```
<html>
<head>
<title>Telephone book search</title>
</head>

<body>
```

```

<cfoutput>
<form action="#request.self#" method="post">
<input type="hidden"
name="#application.fusebox.fuseactionVariable#"
value="#xfa.submitform#">
Search for:
<input type="text" name="name">
<input type="submit" value="Submit">
</form>
</cfoutput>
</body>
</html>

```

We'll rename the 'hello' fuseaction to 'results', so we'll also need to change the value set by xfa.submitform in the welcome fuseaction. This can all be done by editing the circuit.xml.cfm file – see, we're changing our applications flow without editing the individual fuses. We may as well rename our dsp_hello.cfm file to dsp_results.cfm so it's a little bit more meaningful too.

Your circuit.xml.cfm will end up looking pretty similar to;

```

<circuit access="public">

    <fuseaction name="welcome">
        <xfa name="submitform" value="learnFB.results"
/>
        <include template="dsp_welcome.cfm" />
    </fuseaction>

    <fuseaction name="results">
        <include template="dsp_results.cfm" />
    </fuseaction>

</circuit>

```

Now we need to perform a query against our database. Fusebox encourages developers to break down their applications into discrete files which perform various functions. You'll notice that in all the examples so far I've been using a dsp_ prefix in my files that output content, similarly files that perform queries will be named qry_, files and files that perform actions will be named act_.

Our fuseactions have only included a single fuse up till now, but we can stack as many fuses as we like into a fuseaction. They are included in the order they are listed so if you execute a query then reference the recordset the query fuse must come before the display fuse. In a later lesson we'll use another feature of Fusebox4 which allows us to stack fuseactions within fuseactions.

We need to have the form submitted, so we need to run the query before we output results to the user so our results fuseaction ends up looking like;

```
<fuseaction name="results">
    <include template="qry_search.cfm" />
    <include template="dsp_results.cfm" />
</fuseaction>
```

If you haven't already done so, create a file `qry_search.cfm` in your application and open it up for editing.

Our form on `dsp_welcome.cfm` gets the user to enter the name they want to search for in a input box called 'name' so our query will need to use `attributes.name` in it. I need to assume you know how to extract data from db's via SQL, this is after all a lesson in Fusebox and not a lesson in CFMX.

I've gone ahead and written my query to look something like this;

```
<cfquery name="searchresults" datasource="#request.dsn#">
select *
from users
where lastname like '%#attributes.name#%'
</cfquery>
```

Nothing different with that, except now I've added a new variable `request.dsn`. We could just add `request.dsn` to our `Application.cfm` file but this seems like a good opportunity to introduce plugins.

Plugins provide various points in a fusebox request for the developer to run code, that'll become a little clearer in a moment. The idea being, that in previous versions of fusebox for developers to do certain things meant that they needed to customise the core files. With plugins that need is done away with.

There are 6 phases (points) which can be used, `preProcess`, `PreFuseaction`, `PostFuseaction`, `fuseactionException`, `postProcess`, `processError`. In our case, we need a way to load variables into our application before fuseactions have been run, so it makes sense to use a plugin called in the `preProcess` phase.

Fortunately, there are a number of plugins already written for us, one of which will do exactly what we need it to do. If you look in your plugins folder you'll find a file called `globals.cfm`.

If you take a look in `globals.cfm` you'll see it's pretty simple, a variable gets set for the filename to use as the global definition file and then it use a `<CFTRY>` to `cfinclude` the file.

So we now have the plugin we need to add it to our application. If you open up `fusebox.xml.cfm` and scroll down you'll find a block of XML wrapped in `<plugins>` along with the 6 phases you can run code in.

In between the `preProcess` phase add the following line;

```
<plugin name="Globals" template="Globals.cfm"/>
```

Now we've told our app to call the plugin we just need to create a file `myGlobals.cfm` in the root of our application containing variables we want set throughout the application.

```
<cfset request.self = "index.cfm">
<cfset request.myself =
"#request.self?#application.fusebox.fuseactionVariable#"
>
<cfset request.dsn = "learnFb">
```

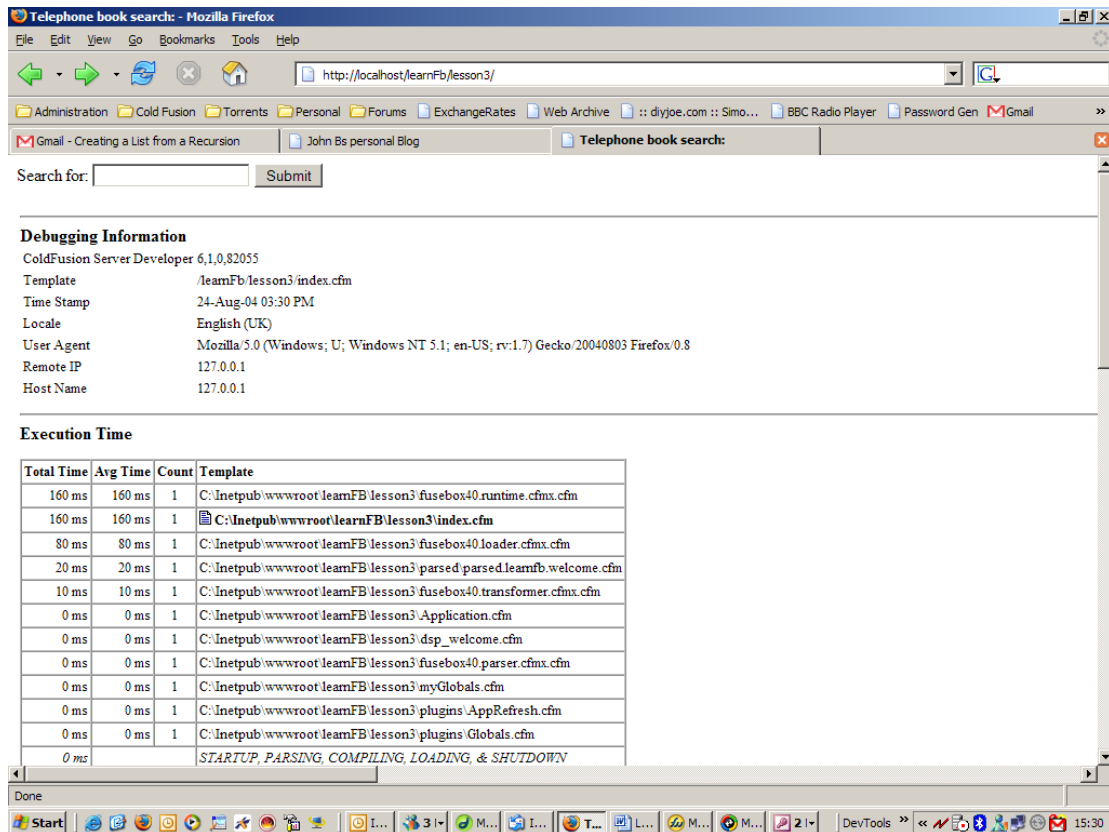
Notice now that since we are within a fusebox request we can use `application.fusebox.fuseactionVariable` in our `request.myself` variable.

So now, all we need do is edit `dsp_results.cfm` to display something back to the user, for now just use the following;

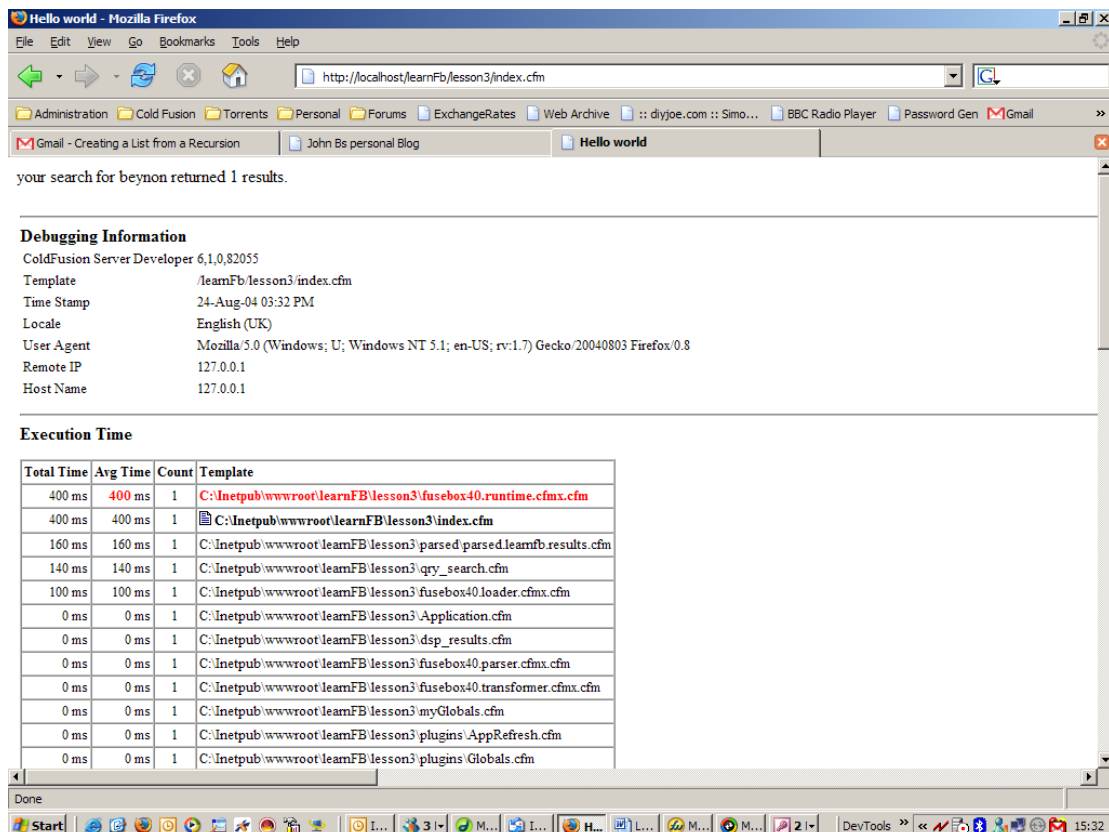
```
<html>
<head>
<title>Hello world</title>
</head>

<body>
<cfoutput>
your search for <i>#attributes.name#</i> returned
#searchresults.recordcount# results.
</cfoutput>
</body>
</html>
```

Go ahead, and test the application.



Our new search for, search for my surname 'beynon'



Message back to user containing their search term and results count.

Let's carry on working on the dsp_results.cfm file to output a table containing the firstname, surname, extension number and department for the results found.

Dsp_results.cfm ends up looking like;

```
<html>
<head>
<title>Hello world</title>
</head>

<body>
<cfoutput>
  <p>your search for <i>#attributes.name#</i> returned
  #searchresults.recordcount# results.
  </p>
  <table width="80%" border="0">
    <tr>
      <td>&nbsp;</td>
      <td><strong>Firstname</strong></td>
      <td><strong>Lastname</strong></td>
      <td><strong>Extension</strong></td>
      <td><strong>Department</strong></td>
    </tr>
    <cfloop query="searchresults">
      <tr>
        <td><a href="##">view</a></td>
        <td>#firstname#</td>
        <td>#lastname#</td>
        <td>#extension#</td>
        <td>#department#</td>
      </tr>
    </cfloop>
  </table>
</cfoutput>
</body>
</html>
```

Nothing unusual in that. Go ahead and rerun the application. If you enter 'bey' you get one result if you enter 'smith' you get two results, if you don't enter anything you get all the results returned.

your search for *smi* returned 2 results.

	Firstname	Lastname	Extension	Department
view	Rachel	Smith	7432	13
view	Fred	Smitham	7116	15

Debugging Information

ColdFusion Server Developer 6,1,0,82055

Template /learnFb/lesson3/index.cfm
 Time Stamp 24-Aug-04 03:43 PM
 Locale English (UK)
 User Agent Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7) Gecko/20040803 Firefox/0.8
 Remote IP 127.0.0.1
 Host Name 127.0.0.1

Execution Time

Total Time	Avg Time	Count	Template
221 ms	221 ms	1	C:\inetpub\wwwroot\learnFB\lesson3\index.cfm
211 ms	211 ms	1	C:\inetpub\wwwroot\learnFB\lesson3\fusebox40\runtime.cfm.cfm
90 ms	90 ms	1	C:\inetpub\wwwroot\learnFB\lesson3\fusebox40\loader.cfm.cfm
71 ms	71 ms	1	C:\inetpub\wwwroot\learnFB\lesson3\parsed\parsed.learnfb.results.cfm
60 ms	60 ms	1	C:\inetpub\wwwroot\learnFB\lesson3\qry_search.cfm
10 ms	10 ms	1	C:\inetpub\wwwroot\learnFB\lesson3\fusebox40\transformer.cfm.cfm
0 ms	0 ms	1	C:\inetpub\wwwroot\learnFB\lesson3\Application.cfm
0 ms	0 ms	1	C:\inetpub\wwwroot\learnFB\lesson3\dsp_results.cfm

Multiple results returned by our db query

At the moment our application only searches on lastname, time to make it a little bit more useable and allow it to search on firstname and department too.

All that involves us doing is editing our qry_search.cfm file to include a few more fields in the where statement,

```
<cfquery name="searchresults" datasource="#request.dsn#">
select userid, lastname, firstname, department, extension
from users
where lastname like '%#attributes.name#%' or firstname
like '%#attributes.name#%' or department like
'#attributes.name#'
</cfquery>
```

If we now run the application we can search on firstname, lastname or a department number. Go try it, search for 'John' and see what comes back then try 96.

Now, let's change our results display file so when results are returned we could click on the department number to see other users in that department. Our query can already handle searching department numbers so we only need to work on the dsp_results.cfm file.

It's a pretty simple modification, change the #department# in the last <td></td> to:

```
<a
href="#request.myself#=#attributes.fuseaction#&name=#department#">#department#</a>
```

Again, we're using the `request.myself` variable, `attributes.fuseaction` is the current fuseaction so we're just sending our variable `name=` into the same fuseaction again to get the results.

Try it out, if you search for 'bey' you'll get one result, if you click on the department link you'll see 2 entries returned.

What if we want to see more information on a specific entry, drilling down onto an entry? For that we'll need a new fuseaction to display data for a specific user.

First off, let's make 'view' a clickable link to a new fuseaction 'details' passing a `userID` into it, as you'd expect we will use an XFA, so our we can change the `` on view to:

```
<a
href="#request.myself#=#xfa.userdetails#&userID=#userID#"
>view</a>
```

This will work as long as we assign a value to `xfa.userdetails` in `circuit.xml.cfm` using the following code on our results fuseaction;

```
<fuseaction name="results">
  <include template="qry_search.cfm" />
  <xfa name="userdetails" value="learnfb.details" />
  <include template="dsp_results.cfm" />
</fuseaction>
```

So that's the results page changed to link to our `details` fuseaction, now we just need to add our new fuseaction to handle the drill down.

Add the following code to `circuit.xml.cfm`

```
<fuseaction name="details">
  <include template="qry_userdetails.cfm" />
  <include template="dsp_userdetails.cfm" />
</fuseaction>
```

Create the two files, `qry_userdetails.cfm`, `dsp_userdetails`.

`Qry_userdetails.cfm` needs to retrieve a specific user record, so go ahead and create that, remember `attributes.userID` will contain the `userID` to retrieve.

```
<cfquery name="userdetails" datasource="#request.dsn#">
select *
from users
where userID = #attributes.userID#
```

```
</cfquery>
```

dsp_userdetails.cfm needs to show all the details of the selected user, returned by the query 'userdetails'

```
<html>
<head>
<title>User Details</title>
</head>

<body>
<cfoutput>
<table width="50%" border="0">
  <tr>
    <td>Firstname</td>
    <td>#userdetails.firstname#</td>
  </tr>
  <tr>
    <td>Lastname</td>
    <td>#userdetails.lastname#</td>
  </tr>
  <tr>
    <td>email address </td>
    <td>#userdetails.emailaddress#</td>
  </tr>
  <tr>
    <td>birth date </td>
    <td>#dateformat(userdetails.birthdate, "dd mmm
yyyy")#</td>
  </tr>
  <tr>
    <td>extension</td>
    <td><p>#userdetails.extension#</p>
    </td>
  </tr>
  <tr>
    <td>department</td>
    <td>#userdetails.department#</td>
  </tr>
</table>
</cfoutput>
</body>
</html>
```

So now, when we click on view next to a specific user we see;

The screenshot shows a Mozilla Firefox browser window displaying the 'User Details' page for user ID 1. The page content is as follows:

User Details

Firstname	John
Lastname	Beynon
email address	john@beynon.org.uk
birth date	08 Feb 1977
extension	7147
department	96

Debugging Information

ColdFusion Server Developer 6,1,0,82055
 Template /learnFb/lesson3/index.cfm
 Time Stamp 24-Aug-04 06:11 PM
 Locale English (UK)
 User Agent Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7) Gecko/20040803 Firefox/0.8
 Remote IP 127.0.0.1
 Host Name 127.0.0.1

Execution Time

Total Time	Avg Time	Count	Template
912 ms	912 ms	1	C:\inetpub\wwwroot\learnFB\lesson3\fusebox40.runtime.cfm.cfm
912 ms	912 ms	1	C:\inetpub\wwwroot\learnFB\lesson3\index.cfm
511 ms	511 ms	1	C:\inetpub\wwwroot\learnFB\lesson3\parsed\parsed.learnfb.details.cfm
491 ms	491 ms	1	C:\inetpub\wwwroot\learnFB\lesson3\qry_userdetails.cfm
181 ms	181 ms	1	C:\inetpub\wwwroot\learnFB\lesson3\fusebox40.loader.cfm.cfm
30 ms	30 ms	1	C:\inetpub\wwwroot\learnFB\lesson3\fusebox40.parser.cfm.cfm

Drilling down to a specific user

Summarizing to this point, we've built a data drill down application to retrieve matching entries from a database of users. We can search on either firstname, lastname or department, drilling down to see full details for a specific user.

If we go back to the last screenshot, we can see the numerical value of userID in the address bar, a URL like that just encourages a user to mess with the value of userID just to see what comes back – cos hey, we know users like to meddle. All well and good if there are matching records, but if there aren't they just see an empty table.

We've got a number of options of how to handle a query returning a record count of zero,

1. we could wrap the table in dsp_userdetails.cfm within a `<cfif userdetails.recordcount>` block with a `<cfelse>` to display a message back to the user.
2. use conditional logic somewhere to handle a recordcount of 0 and redirect user accordingly

In Fusebox4 we are able to use basic conditional logic within the circuit.xml.cfm file using an `<IF></IF>` block. So we could go ahead and write our details fuseaction now as:

```
<fuseaction name="details">
```

```

<include template="qry_userdetails.cfm" />
<if condition="userdetails.recordcount">
    <true>
        <include template="dsp_userdetails.cfm" />
    </true>
    <false>
        <relocate url="#request.self#" />
    </false>
</if>
</fuseaction>

```

Let's look a little closer, we've run our query in `qry_userdetails.cfm` and then we testing the condition `userdetails.recordcount` in the `<IF condition>` block. If the query `userdetails` returns results then the condition is true, if it doesn't return results it will be false. If it is true, we go ahead and include our `dsp_userdetails.cfm` file to show the results to the user. If it's false, we send the user back to the start of the application – serves the user right for trying to mess our app up! By default the `<relocate>` performs a client side redirect, if we were to add `type=server` we can perform a server side redirect also.

NOTE: We can't nest logic, it is basic 1 level logic at the `circuit.xml.cfm` level, if you need deeper you will have to use a fuse to perform the logic for you.

Phew, that was a long lesson – time to grab a coffee. In this lesson, we've covered handling form submissions; we've turned our hello world! Application into a real world application, a telephone search directory. We used multiple fuses in a `fuseaction` and introduced basic conditional logic into our circuit definition file along with client or server side redirection, would you believe – we've barely touched the surface of Fusebox4 yet 😊