Getting Started with PHP Fusebox

By Mike Britton

All materials Copyright © 1997−2002 Developer Shed, Inc. except where otherwise noted.

Table of Contents

Introduction

The Fusebox web methodology may change the way you approach architecting data−driven web applications.

The latest version of Fusebox is the most scalable and effective incarnation of a "web box" − style architecting approach. Created by Steve Nelson, Fusebox has been further popularized by the arrival of a PHP port by David Huyck of http://bombusbee.com. Armed with Fusebox and a few simple tools, PHP developers can now architect scalable web applications across project teams, technical disciplines, and display mediums.

If you've ever found yourself working long hours trying to debug code in an application that you know will probably not be appreciated by the client, you're not alone. The fact is that **most software development projects fail.** That's right, and the number is scary: 70%. They fail for a number of reasons − excessive time spent coding and too little spent understanding the client's needs, poor documentation, code that can't be reused, code that's difficult to maintain. The list goes on and on.

You now have the opportunity to find out why using Fusebox can make your life much easier by dramatically increasing your project's chances of success. Based on a hub−and−spoke model, Fusebox allows sections of code to work independently of each other while promoting code reuse, distributed development and cheaper maintenance. This translates to a happy client and happy developers − the perfect end to any project.

Let's jump into the code and set up a simple application. As we build, the possibilities will reveal themselves, and hopefully we'll emerge on the other side with a handy technique for your web−building arsenal.

Step 1: Setting Up the Core Files

If you have access to a web server running PHP, you have all you need to begin using PHP Fusebox. Well, almost.

We're going to download the PHP Fusebox core files, and then learn how they work. Then we'll set up two circuits. In this case it'll be a simple website with two sections: **home** and **biography**.

Now let's set up our application!

First you need to download and copy the **core files** into the root directory of your application. These files can be found at http://sourceforge.net/projects/php−fusebox/ in their pristine state, ready to be shaped into a new website.

The Core Files:

index.php

DefaultLayout.php

dsp_main.php

fbx_Circuits.php

fbx_Fusebox3.0_PHP4.0.6.php

fbx_Fusebox3.0_PHP4.1.x.php

fbx_Layouts.php

fbx_Settings.php

fbx_Switch.php

fbx_ListFunctions.php

fbx_SaveContent.php

What do the "core files" do?

Let's go take a look at these files now. If there are any I don't cover today, we'll be going over them in a future article.

fbx_Circuits.php

Included by fbx_Fusebox3.0_PHP4.1.x.php (see below), *fbx_Circuits.php* is where the circuits of our Fusebox are defined. This is the place where values are set that determine the structure of the app. A typical *fbx_Circuits* (and the one you downloaded) looks like this:

```
<?

$Fusebox["circuits"]["home"] = "home";

?>
```

Don't worry − we'll be adding circuits to this code soon enough.

fbx_Fusebox3.0_PHP4.1.x.php (or fbx_Fusebox3.0_PHP4.0.6.php)

This is the "engine" of the Fusebox application − the file that makes it all possible. It does the following things:

1. Includes *fbx_Circuits.php*, where the *circuits* array is populated with your application's circuit definitions. The circuit definitions in *fbx_Circuits.php* map your site's folder structure to a central *circuits* structure.
2. Creates a reverse−lookup of the directory structure you defined in *fbx_Circuits.php.*
3. Includes *fbx_Settings.php,* where default values are set for your application.
4. Receives the *fuseaction* and *circuit* variables, which control what content is displayed in your application.
5. Executes the *fuseaction* in the correct circuit's fbx_Switch file.
6. Includes *fbx_Layouts.php,* where you can customize the look and feel of your application.

You'll be happy to know that this file should not be modified in any way, and can function as the heart of your web application right out of the box, so to speak. You should understand the full functionality behind this key file, but this is by no means required since editing the file's source code could make your application incompliant with the Fusebox 3 specification, defeating the whole purpose of using the methodology in the first place!

fbx_Switch.php

This file consists of a switch/case statement that tells the application which files to include. Here's the code you see when you open it up:

```
switch($Fusebox["fuseaction"]) {

    case "main":
```

```
    case "Fusebox.defaultFuseaction":

        include("dsp_main.php");

        break;

    default:

        print "I received a fuseaction called <b>'" . $Fusebox["fuseaction"] . "'</b> that circuit <b>'" .
$Fusebox["circuit"] . "'</b> does not have a handler for.";

        break;

}
```

Leave this file alone for now. We'll be back to it shortly.

fbx_Settings.php

This is where default values are set for each circuit. Circuits (subdirectories) each have their own versions of this file. If they do, the variables' values in the individual circuits' *fbx_Settings.php* files will overwrite their default values in the root directory's *fbx_Settings.php.*

index.php − *or whatever your server's default document is called.*

This is where we load the core file, or for lack of a better word *engine*, of the Fusebox architecture: *fbx_Fusebox3.0_PHP4.1.x.php*. We'll be linking back to this file for every page request and passing it a variable called a *fuseaction*. The *fuseaction* will give the engine the information it needs to display the circuit's content. Note: this file can be renamed to whatever your web server uses as its default document.

DefaultLayout.php

This is a typical layout file, and is required in each circuit of your application. If you were making an application with many circuits, you'd want to make sure this file is placed in each circuit's directory, along with the other core files required in each circuit. In a future article we'll discuss the incredibly cool use of *nested layouts*, which are another example of the versatility and scalability of Fusebox−based architecting.

fbx_Layouts.php

You should have at least one *fbx_layouts.php* in your application, depending on which circuits require unique layouts. This file controls the layout of each circuit or $Fusebox["layoutFile"] element. Any circuit that has its own layout requirements should have its own *fbx_Layouts.php* file in its root directory. If not, *fbx_Layouts.php* can be omitted and the application root's *fbx_Layouts.php* will take over. Fusebox enables you to "nest" these layouts in a powerful way, but that topic is best left for another time.

dsp_main.php

A typical display file. This file will be included in our main circuit.

A Word on FuseDocs

Requirements baselines and prototyping are thoroughly embedded in the architecting techniques favored by Fusebox developers. Part of this process is the creation of documentation elements called *FuseDocs.* The basic form of a Fusebox 3−compliant XML fusedoc for PHP:

```
/*

<fusedoc fuse="dsp_circuitName.php">

    <responsibilities>

        I display information to the user.

    </responsibilities>

    <history author="Mike Britton" email="mbritton72@djtrock.com" date="20 April 2002"
type="Update">File created</history>

    <io>

        <in>

<string name=" " default="" />

<string name=" " default="" />

</in>

<out>

<boolean name=" " default=" " />

<string name=" " default="" />

        </out>

    </io>

</fusedoc>

*/
```

For more on FuseDocs, visit http://www.HalHelms.com, where you'll find a wealth of resources on the subject.

Fusebox Naming Conventions

You're trying to make a website, so you'll obviously need content. The files that make up your Fusebox application are broken down into a few different categories:

fbx_xxx.php

The Fusebox "core" files, or files that act together to provide the Fusebox functionality.

dsp_xxx.php

Display fuses, prefixed with "dsp", are used to show information to the user.

qry_xxx.php

Query fuses, prefixed with "qry", contain all your database queries.

act_xxx.php

Action fuses, prefixed with "act", typically perform some kind of action, such as emailing someone with the mail() function.

url_xxx.php

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

Keeping the responsibilities of each type of file in mind, it should become apparent how to properly fashion your FuseDocs. A good rule of thumb is to make sure a developer who knows nothing about the overall application can code each fuse based entirely on the fusedoc.

Picking Up Where We Left Off: Setting Up the Core Files

Now that we've copied the core files into the root directory, let's go ahead and create a simple circuit app that demonstrates the basic principles behind the PHP Fusebox architecture.

1. Open fbx_Circuits.php. You'll see the fusedoc XML at the top, followed by the default home circuit definition:

```
$Fusebox["circuits"]["home"] = "home";
```

This code is identifying the $Fusebox["circuits"]["home"] nested array element as "home". The lack of slash mark(s) in the value in quotation marks tells us this circuit is in the root directory.

When the $Fusebox["circuits"] structure is passed, *fuseaction=home.main* in the URL for instance, the application will know this circuit is in the application root, and the core "engine" file, *fbx_Fusebox3.0_PHP4.1.x.php*, will include the application root's *fbx_Switch.php* to complete the request.

Note: Although it isn't explicitly required to create PHP Fusebox 3 web sites, you can see how all this works by opening the "engine" file we discussed earlier, *fbx_Fusebox3.0_PHP4.1.x.php.* Checking under the hood of this file is advisable. The code is well commented with in−line comments and FuseDocs.

Ordinarily, if we were constructing a complex web application with PHP Fusebox, we'd add additional code to define more circuits in the application root's *fbx_Circuits.php:*

```
$Fusebox["circuits"]["sectionOne"] = "home/sectionOne";

$Fusebox["circuits"]["sectionTwo"] = "home/sectionTwo";
```

There could be as many circuits as you want. For our purposes, we'll only be working with two circuits: the "home" and "biography" circuits. Change the code of *fbx_Circuits.php* to read:

```
$Fusebox["circuits"]["home"] = "home";

$Fusebox["circuits"]["biography"] = "home/bio";
```

The circuit you have just defined below the "home" circuit could be accessed through either the URL or some kind of form action (POST or GET). For our purposes, we'll be calling our circuits through simple text links.

2. In the root directory of your application, open fbx_Switch.php. This is where you'll define all the fuses in this, the home (or root) circuit. Like I said before, you'll be using the switch / case statement in this file to include files based on the fuseaction passed to the application.

The default code looks like this:

```
switch($Fusebox["fuseaction"]) {

    case "main":
```

```
    case "Fusebox.defaultFuseaction":

        include("dsp_main.php");

        break;

    default:

        print "I received a fuseaction called <b>'" . $Fusebox["fuseaction"] . "'</b> that circuit <b>'" .
$Fusebox["circuit"] . "'</b> does not have a handler for.";

        break;

}
```

The switch statement switch($Fusebox["fuseaction"]) receives the $Fusebox["fuseaction"] variable from the URL, a form ñ− whatever you want to use to pass it − and passes the fuseaction down through the case statements until it finds one that matches. When a match is encountered, the matching case statement specifies which files are included. In this situation, if the **"main"** fuseaction were specified, the file *dsp_main.php* would be included. If no fuseaction were passed, the same file would be included because the default case statement calls the same fuse.

Using XFAs

Let's say you want one screen on your web site to lead into another, like in a traditional login system where the user types his / her username and password into a text field and presses the submit button. In Fusebox, instead of hard−coding a path to a CGI script or another PHP page in your form's "action" parameter, you simply specify an *exit fuseaction*, or **XFA**.

What's an XFA?

PHP Fusebox filenames must begin with either ***dsp_, act_, url_,*** or ***qry_.*** They contain *exit fuseactions* that are picked up in a circuit's *fbx_Switch.php* file.

For example, one of your circuits' dynamically−included files will contain references collectively known as XFAs. When called, these XFAs communicate with the switch/case statement in *fbx_Switch.php,* telling it to find a matching circuit (case statement), and an XFA definition inside the case statement, one that tells the application where the **exit** for that fuseaction is. It helps to remember "X for exit".

To further illustrate, let's focus again on *fbx_Switch.php.* Add the following code under the initial case statements:

```
$XFA["biography"] = "biography.hello_world";
```

Now your switch statement should look like *this:*

```
switch($Fusebox["fuseaction"]) {

    case "main":

    case "Fusebox.defaultFuseaction":

// Add this line:

$XFA["biography"] = "biography.hello_world";

        include("dsp_main.php");

        break;

    default:

        print "I received a fuseaction called <b>'" . $Fusebox["fuseaction"] . "'</b> that circuit <b>'" .
$Fusebox["circuit"] . "'</b> does not have a handler for.";

        break;

}
```

This may take a while to get right in your head. While *fbx_Switch.php* dynamically includes files, it also controls the flow of the application. Think of it as a railroad switch that directs your model train by moving the switch to the correct track. Your train signals the switch where it wants to go, and the switch moves into place to match the signal sent by the train. What you've just done is instruct the circuit to go to the **bio/** directory (or "biography" circuit if you think about it from the perspective of the application root's *fbx_Circuits.php* file) when passed an XFA called "biography". The biography circuit's *fbx_Switch.php* will have a case statement that looks for the fuseaction "hello_world" and include the files you define for this circuit.

But wait, we're getting ahead of ourselves. Deeeep breath. First, we have to understand how these XFAs are passed in the context of the application.

An XFA or *exit fuseaction* called from a URL:

```
<a href=\"$PHP_SELF?fuseaction=".$XFA["biography"]."\">Example Link</a>
```

An XFA or *exit fuseaction* called from form POST:

```
<form action=\"$PHP_SELF?fuseaction=".$XFA["biography"]."\" method=\"post\">
```

Or a GET:

```
<form action=\"$PHP_SELF\">

<input type=\"hidden\" name=\"fuseaction\" value=\"".$XFA["biography"]."\">

</form>
```

The beauty of PHP Fusebox is in the abstraction of the file system's real structure: using XFAs, you can send the application to any circuit (directory) you want, and have it perform whatever duties you wish. As long as the root directory's *fbx_Circuits.php* contains a matching circuit definition and its *fbx_Switch.php* contains a corresponding XFA that can tell *fbx_Circuits.php* where to go next, the sky's the limit!

Step 2: Creating the "Biography" Circuit

You've added code that references the biography circuit, so now it's time to create it! In your application root (the one you're calling "home" in *fbx_Circuits.php)*, create another directory and call it **bio.** This is where we're going to build the biography circuit.

Now you must copy the following PHP Fusebox core files into the bio directory (it's a subset of the same files found in your application root):

DefaultLayout.php

dsp_main.php

fbx_SaveContent.php

fbx_Settings.php

fbx_Switch.php

index.php

See, there's *fbx_Switch.php* again! The biography circuit's *fbx_Switch.php* will include the files needed to satisfy the fuseactions passed to it, as we've already discussed.

Open the /bio directory's *fbx_Switch.php.*

You will see similar code to the *fbx_Switch.php* we already edited, in the application root:

```
switch($Fusebox["fuseaction"]) {

    case "main":

    case "Fusebox.defaultFuseaction":

        include("dsp_main.php");

        break;

    default:

        print "I received a fuseaction called <b>'" . $Fusebox["fuseaction"] . "'</b> that circuit <b>'" .
$Fusebox["circuit"] . "'</b> does not have a handler for.";

        break;

}
```

Keep this file open, and focus back on *fbx_Switch.php* in the application root. Remember when we created an XFA for the "**biography**" circuit?

```
switch($Fusebox["fuseaction"]) {

    case "main":

    case "Fusebox.defaultFuseaction":

// Add this line:

$XFA["biography"] = "biography.hello_world";

        include("dsp_main.php");

        break;

    default:

        print "I received a fuseaction called <b>'" . $Fusebox["fuseaction"] . "'</b> that circuit <b>'" .
$Fusebox["circuit"] . "'</b> does not have a handler for.";

        break;

}
```

Now we're going to adjust the *fbx_Switch.php* file in **/bio** to accommodate the **XFA** we just set up in the application root's *fbx_Switch.php*. Add the code in **bold** to the /bio directory's *fbx_Switch.php*:

```
switch($Fusebox["fuseaction"]) {

    case "main":

    case "Fusebox.defaultFuseaction":

        include("dsp_main.php");

        break;

    case "hello_world":

        include("dsp_hello_world.php");

        break;

    default:

        print "I received a fuseaction called <b>'" . $Fusebox["fuseaction"] . "'</b> that circuit <b>'" .
$Fusebox["circuit"] . "'</b> does not have a handler for.";
```

```
        break;

}
```

Now open *dsp_main.php* in **/bio** and save it back down as *dsp_hello_world.php.* Add the following code to *dsp_hello_world.php:*

```
<h3>Biography: Hello World!</h3>

This is the place where the world can come learn about me.

<p>

<?

echo "<a href=\"$PHP_SELF?fuseaction=".$XFA["home"]."\">Home</a>";

?>
```

Almost There!

Now that both *dsp_* pages are there for either circuit, the fusebox should work, right? It will − as soon as you add the XFA you just referenced in *dsp_hello_world.php* to the *fbx_Switch.php* in **/bio.** Without reference to this XFA in your *fbx_Switch.php file*, your PHP Fusebox application won't know what you're talking about!

Open the *fbx_Switch.php* in /bio and add the line in bold.

```
switch($Fusebox["fuseaction"]) {

    case "main":

    case "Fusebox.defaultFuseaction":

        include("dsp_main.php");

        break;

    case "hello_world":

        // Add this line:

$XFA["home"] = "home.main";

        include("dsp_hello_world.php");

        break;

    default:

        print "I received a fuseaction called <b>'" . $Fusebox["fuseaction"] . "'</b> that circuit <b>'" .
$Fusebox["circuit"] . "'</b> does not have a handler for.";

        break;

}
```

As soon as you do this, add an XFA link to the biography circuit in the application root's *dsp_main.php*, and you'll be finished with your first PHP Fusebox application!

```
<h3>Home: Main Page</h3>

This is the home page.  Click the link below to find out more about me.

<p>

<?
```

```
echo "<a href=\"$PHP_SELF?fuseaction=".$XFA["biography"]."\">Biography</a>";

?>
```

Now fire up a browser and surf to your application root. The default fuseaction should be triggered automatically since you didn't type a fuseaction.

Mouse over the link to go to **Biography.** Notice how the URL is now a real fuseaction and not an XFA? That's because we defined the XFA in the application root's *fbx_Switch.php* file. The Fusebox knows this XFA is a call to the application root's **$Fusebox["circuits"]["biography"]** circuit, which you can see when you open *fbx_Circuits.php* in the root:

```
$Fusebox["circuits"]["home"] = "home";

$Fusebox["circuits"]["biography"] = "home/bio";
```

Now click on the **Biography** link. The Biography pages comes up. Now click the **Home** link. You're back home. Continue doing this for a few hours while listening to high−energy dance music!

If you think this might work for you, don't stop here. There are great Fusebox−related sites all over the web. And Fusebox isn't just for PHP − it was originally written for ColdFusion and is available for Java, as well.

PHP−Fusebox Links

David Huyck's Bombusbee:

http://bombusbee.com

Fusebox for PHP at SourceForge:

http://sourceforge.net/projects/php−fusebox/

Wireframing tool for PHP Fusebox:

http://php−fusebox.sourceforge.net/index.php?fuseaction=downloads.main

Demo Applications:

http://php−fusebox.sourceforge.net/index.php?fuseaction=downloads.main

Fusebox Links

Fusebox.org:

http://www.fusebox.org

Steve Nelson's SecretAgents.com:

http://www.secretagents.com

Hal Helms' Personal Site:

http://www.halhelms.com

Lee Borkman's Bjork.net:

http://www.bjork.net