

## Class Library

This documentation refers to Paul Clement's Class Library, which can be downloaded from the subversion repository at <http://software.clempaul.me.uk/svn/classlib/>

The class library is written in PHP and can be used in any project as long as the copyright notice is left intact.

Please do not redistribute any of the files that make up this class library. The latest version will always be available from the subversion repository listed above.

## Database Abstraction Class

The Database Abstraction Class (db.class.php) allows easier data access with the MySQL Database Server.

To initialise the class, you must first include the library file and then use the following code to initialise the database connection:

```
$DB = new DataBase('Host', 'User', 'Password', 'Database Name');
```

### Accessing Data

Accessing data that is already in the database is easy, with the following syntax:

```
$DataSet = $DB->GetDataSet("SQL Command");
```

This will return all of the rows that are returned from the SQL query in an array. It is possible to loop through the rows that are returned using a foreach statement e.g.:

```
foreach($DB->GetDataSet("SELECT * FROM test") as $ColumnName, $Value)
```

It is also possible to only return a single row from the database using the GetDataRow command. This returns the first row from the SQL query, which can be accessed directly. The syntax is:

```
$DataRow = $DB->GetDataRow("SQL Command");
```

### Inserting Data

Inserting data to a database table is also easier. The data that is to be stored in the database can be put into an array, which is then passed to the InsertRow command. E.g.:

```
$Data = array ( "ColumnName" => "Value", "Column2" => "Value2");
```

```
$DB->InsertRow("TableName", $Data);
```

### Updating Data

It is just as easy to update data that is already in the database. 2 arrays are passed to the UpdateRow command, one for the data and one for the WHERE clause. E.g.:

```
$Data = array("Column1" => "Value1", "Column2" => "Value2");
```

```
$Where = array("PrimaryKey" => "Value");
```

```
$DB->UpdateRow("TableName", $Data, $Where);
```

### Deleting Rows

Deleting rows of data that are already in the database is also very easy. The WHERE clause is put into an array which is then passed to the DeleteRow command. E.g.:

```
$Where = array("PrimaryKey" => "Value");
```

```
$DB->DeleteRow("TableName", $Where);
```

### **Executing SQL**

If you wish to perform some complex or obscure SQL command, this can be done by using the following syntax:

```
$DB->ExecuteSQL("SQL Command");
```

## Master Pages

The MasterPage class (masterpage.class.php) allows a file containing HTML for an entire page to have content areas added which can be filled from PHP code. This removes the need for using include to import HTML around your own code.

The files that can be consumed using the class can have three types of additional HTML tags added to build in this functionality. These are:

`<master:PageTitle />` which is used to represent a sub title for a page.

`<master:ContentArea id="ID" />` where ID is a unique identifier which is used to add content to the page. The ID must consist only of A-Z, a-z, 0-9, - and \_. Use of other characters may produce errors.

`<master:HeadData />` which is used to add additional information such as external JavaScript, CSS files and RSS feeds to the page.

An example Master Page file is shown below:

```
<html>
<head>
<title>Site Name<master:PageTitle /></title>
<master:HeadData />
</head>
<body>
<master:ContentArea id="Area1" />
</body>
</html>
```

## Initialisation

After you have included the class file you can initialise the class using the following syntax:

```
$Master = new MasterPage("MasterPage.master");
```

Where MasterPage.master is the location of your master page file.

## Page Titles

You can fill `<master:PageTitle />` tags to add sub titles to pages using the following syntax:

```
$Master->SetPageTitle("Page Title", "-");
```

The second parameter is optional and defines the separator between the main title and the sub title. The default is a bullet point.

## Filling Content Areas

Filling content areas is done by putting the content for the area into a string and passing it to the FillContentArea method. E.g.:

```
$Master->FillContentArea("AreaID", $Content);
```

It is sometimes easier to use [Output Buffering](#) to create the string.

### Adding data to the Head

These methods all place additional strings into the <master:HeadData /> tag.

JavaScript files can be included by passing the URL of the script file to the AddJavaScript method:

```
$Master->AddJavaScript("URL");
```

CSS files can be included by passing the URL of the CSS file to the AddCSS method:

```
$Master->AddCSS("URL");
```

RSS feeds can be linked to by passing the Title and URL of the feed to the AddRSS method:

```
$Master->AddRSS("URL", "Title");
```

Other data (such as meta tags) can be added by passing specific HTML to the AddHeadData method:

```
$Master->AddHeadData("<meta http-equiv='content-type' content='text/html; charset=utf-8' />");
```

### Displaying the Page

Once you have finished assembling your page using the above methods, you can display it by using the Display method e.g.:

```
$Master->Display();
```

You can also get the content put into a string for additional processing, using the ToString method e.g.:

```
$Master->ToString();
```

### Nested Master Pages

Although you cannot truly nest master pages, you can use multiple instances of the class to manage multiple master pages at one time, and then use the ToString method to fill a Content Area in a different master page with the content from another. E.g.:

```
$Master = new MasterPage("1.master");
```

```
$Sub = new MasterPage("2.master");
```

```
$Sub->FillContentArea("Area1", "Some HTML");
```

```
$Master->FillContentArea("MasterPageArea", $Sub->ToString());
```