G-Forces Development Standards

Version 3.2

February 2008

G-Forces Web Management Ltd
Essendine House
71 College Road
Maidstone
ME15 6SX

T: +44 (0)1622 609707
F: +44 (0)1622 609511
E: info@gforces.co.uk
W: www.gforces.co.uk

Contents

1 Revision History

Revision	Change Log	Author	Date
0.1 – 0.2	Initial Revision	Tim Pickup	15/05/05
0.3	Add – 1. Revision History Edit – 2.1 Naming Conventions : Clearer examples Edit – 2.1.4 Private/Protected Methods and Variables : Minor change Edit – 2.4 Function Definitions : Using PEAR style bracers Moved – 2.9.1 to 4 Appendix – phpdoc Edit – 3 Database : changed naming scheme, clarified some other issues	Tim Pickup	26/05/05
1.0	Insert – 2.1 Folder Structure		
2.0	Overall of processes and guidelines	Barry Last, Colin Shipton, Samuel Smithson	21/10/05
2.0.1	Add 4.12 Address tag	Colin Shipton	26/10/05
2.0.2	Amend 6.1 Database names	Colin Shipton	01/11/05
2.0.3	Add 9.1 Uploads	Colin Shipton	03/11/05
2.0.4	Amend 6.1 Database names Amend 6.2 Table names Amend 6.3 Field names	Colin Shipton	05/01/06
2.0.5	Add 6.6 Administration User Table	Colin Shipton	26/01/06
2.0.6	Added 4.12 Email Confirmation moved 4.12 to 4.13	Barry Last	09/03/2006
3.0	Full Update to all standards	Barry Last	01/02/2007
3.1			
3.2	Updated the PHP class structure and coding standards	Mike Davis	15/02/2008

2 Design

All G-Forces' websites and applications must meet the requirements listed below. Any deviation from this set of rules is not acceptable unless clearly stated in writing. If you need clarification on any point listed, please speak to the Project Team.

2.1 Search Engines

All websites must be optimised for search engines unless clearly stated in writing. This therefore means adhering to a number of design rules:

- Navigation must primarily be, textually based
- All links on the home page of a website should be textually based
- Space must be allocated on home page to allow for 300 words (can be spread over page)
- All titles on a website must be text based

2.2 Screen Resolution & Browsers

All applications and websites must be designed to work in screen resolutions of 1024x768 and above unless clearly stated in writing.

All applications and websites must be optimised for web-browsers that support current web standards (i.e. Internet Explorer 6.0+, Mozilla Firefox 1.0+ and Safari 2.0).

2.3 Fonts & Colours

Consideration must be given to the size and colour of fonts used – any site that fails to meet satisfactory compliance with the customer will require re-building out of office hours. Unless specified the default font colour should be #000000 or #333333. **For more details see CSS 3.1**

2.4 File Size & Downloads

All applications and websites must be optimized for accepted download speeds unless clearly stated in writing.

2.5 Miscellaneous

All application and website designs must be signed off prior to Style Sheets being built. The project team must have written confirmation (e-mail is acceptable) from the customer that the design is accepted.

The use of internally scrolling pages should only be used where all other design possibilities have been exhausted.

If appropriate a favicon.ico should be created an applied to the initial layups.

2.6 Client Area

All new projects my be placed on http://clients.gforces.co.uk. The template for this section can be found in the Creative/layups folder in the base G-Forces Client Folder Template. Links to layups, XHTML templates, wireframes and working site must be present when each section is completed. For example:

- Layups
- XHTML Templates
- Wireframes
- Working site

3 Creative

3.1 CSS Standards

- All websites should be styled to display correctly in Mozilla Firefox as this is the closest standards compliant browser. Conditional statements and separate stylesheets should then be used for Internet Explorer 6 and 7. Below is an example conditional statement:

```
<!--[if IE 7]>
        <link rel="stylesheet" href="ie7.css" media="screen,projection" type="text/css" />
<![endif]-->
```

- Websites must be developed with a printer friendly style sheets.
- CSS will be checked to be compliant with:
  - Internet Explorer 6.0 and 7.0
  - Mozilla Firefox 1.0+
  - Safari 2.0
- All menus must be text based unless stated in writing
- Font Sizes will be resizable as a percentage (%) or em not fixed pixels
- Preferred fonts are font: 62.5%/1.6em Verdana, Arial, Helvetica, sans-serif;
- CSS will be included as a separate file for caching (not inline or at top of page)
- Headings will be written with <h1, h2, h3 etc. – not classed <p> tags
- Headings will represent structural hierarchy and not be abused for design
- CSS will format existing HTML tags where possible – (not classes) (i.e. <h1><p> etc…)
- Physical addresses must be enclosed within <address> tags to facilitate localised searching from search engines.

3.2 Design/Layup Standards

- Sites will, where necessary use breadcrumbs to show users where they are in the site.
- Design resizable up to 1024 x 768 – Adobe Photoshop document size should be 960px wide.
- Design will be navigable on a Mono-screen (print screen & change to Mono in Photoshop)
- All menus must be text based
- Do not use images of dimensions: 468x60 (banner ad sizes)
- Site map to be include in layup where necessary

3.3 Validation

- W3C (XHTML & CSS)
- WCAG 1.0 AA

3.4 Development Standards

- All paths & file names will be created and linked to in camelCase e.g. testPageFile.html.
- For physical addresses of clients offices, use the <address></address> tag
- Use a <label> tag around all text that accompanies a form element – i.e.:

```
<LABEL for="firstname">First name: <INPUT type="text" id="firstname"
tabindex="1"></LABEL>
```

- Relative paths will be calculated in server-side coding to minimise 404 errors
- Forms with image submit buttons must be “<input type=image” not <img onclick=””
- All forms must have a “proper” submit button (even if that is a 1px shim). This is needed for audio browsers.
- All pages must include meta title, meta keywords & description (editable via CMS – CMS overwrites default values)
- Where hyperlinks open in a new window the hyperlink must have a title that says “This page will open in a new window”
- Do not use “spam” or “ad” words within the site structure or file names (unless the content is an advert). Spam words include:

- Banner
- Ad
- Popup
- Advert
- Do not use “spam” or “ad” words within the site structure or file names (unless the content is an advert). Spam words include:

4 XHTML

4.1 XHTML !DOCTYPE

All XHTML pages must include the correct !DOCTYPE declaration to validate as correct XHTML. Browsers will treat your document differently depending on the <!DOCTYPE> declaration. If the browser reads a document with a DOCTYPE, it might treat the document as "correct". Malformed XHTML might fall over and display differently than without a DOCTYPE.

The following !DOCTYPE must be used:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

4.2 XHTML NameSpace

All XHTML pages must include the correct NameSpace declaration directly below the !DOCTYPE declaration to correctly identify the XHTML to the browser.

The following NameSpace must be used:

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
```

4.3 Title tag

All pages must include a <title> tag which is used to place the correct information in the title bar and also helps with search engine optimization, the title tag must be the first tag in the <head> of the document.

4.4 Content-Type

All XHMTL pages must include the correct content-type declaration in the <head> of the document directly underneath the <title> tag to correctly identify the correct character set to the browser.

The following content-type must be used:

```
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
```

4.5 Lower Case Tag And Attribute Names

XHTML is case sensitive, and XHTML only accepts lower case HTML tags and attribute names, therefore all tag and attribute names must be in lowercase. For example:

Incorrect example:

```
<INPUT TYPE=”text”
```

Correct example:

```
<input type=”text”
```

4.6 Quoted Attributes

All XHTML attribute values must be quoted. For example:

Incorrect example:

```
<input type=text
```

Correct example:

```
<input type=”text”
```

4.7 Empty Tags

Empty tags are not allowed in XHTML. The <hr> and
 tags must be replaced with <hr /> and
. All other empty tags such as <img>, <meta>, etc. must also be closed with /> at the end of the tag. For example:

Incorrect example:

```
<img src="../images/layup/spacer.gif">
```

Correct example:

```
<img src="../images/layup/spacer.gif" />
```

4.8 Attribute Minimization

Attribute minimization is forbidden in XHTML. For example:

Incorrect examples:

```
<input checked>
<input readonly>
<input disabled>
<option selected>
```

Correct examples:

```
<input checked="checked" />
<input readonly="readonly" />
<input disabled="disabled" />
<option selected="selected" />
```

4.9 ID Attribute Replaces NAME Attribute

HTML 4.01 defines a name attribute for the elements a, img, and map. In XHTML the name attribute is deprecated, use id instead. For example:

Incorrect example:

```
<img src="picture.gif" name="picture1" />
```

Correct example:

```
<img src="picture.gif" id="picture1" />
```

4.10 Deprecated Elements

Elements that have been deprecated in the most recent version of XHTML include:

4.10.1 Tags

- <applet></applet> - The applet element has been deprecated in favor of <object></object>.
- - The b element has been deprecated in favor of or style sheets.
- <basefont /> - The basefont element has been deprecated in favor of style sheets.
- <blackface></blackface> - The blackface element is a WebTV (MSN TV) element that has been deprecated in favor of style sheets.
- <blockquote></blockquote> - The blockquote element is deprecated in favor of style sheets when used to indent text.
- - The center element has been deprecated in favor of style sheets.
- <dir></dir> - The dir element has been deprecated in favor of .
- <embed /> - The embed element has been deprecated in favor of <object></object>.
- - The font element has been deprecated in favor of style sheets.
- - The i element has been deprecated in favor of or style sheets.
- <isindex /> - The isindex element has been deprecated in favor of the input element and CGI forms.
- <layer></layer> - The layer element has been deprecated in favor of style sheets.
- <menu></menu> - The menu element has been deprecated in favor of .
- <noembed></noembed> - The noembed element has been deprecated in favor of <object></object>.
- <s></s> - The s element has been deprecated in favor of style sheets.
- <shadow></shadow> - The shadow element has been deprecated in favor of style sheets.
- <strike></strike> - The strike element has been deprecated in favor of style sheets.
- <u></u> - The strike element has been deprecated in favor of style sheets.

4.10.2 Attributes

- Alink - The alink attribute defines the color of an active link on the Web page. It has been deprecated in favor of style sheets.
- Align - Used to align elements vertically and horizontally on the document page, the align attribute has been deprecated in favor of style sheets.
- Background - The background attribute is used to define a background image for the element. It has been deprecated in favor of style sheets.
- Border - Used to define a border around the element, the border attribute has been deprecated in favor of style sheets.
- Color - Used to define the color of the enclosed text, the color attribute has been deprecated in favor of style sheets.
- Compact - The compact attribute is used in list tags to create a list that takes up less space. It has been deprecated in favor of style sheets.
- Face - Used to define the font face of the enclosed text, the face attribute has been deprecated in favor of style sheets.
- Height - Used to define the height of the element, the height attribute has been deprecated in favor of style sheets.
- Language - The language attribute is used to define the language used by the script element. It has been deprecated in favor of the type attribute.
- Link - The link attribute defines the color of a link on the Web page. It has been deprecated in favor of style sheets.
- Name - The name attribute names the element for use with dynamic content, it has been deprecated in favor of the id attribute.
- Noshade - The noshade attribute removes the 3-D effect from horizontal rules. It has been deprecated in favor of style sheets.

- Nowrap - Used to stop table contents from wrapping, the nowrap attribute has been deprecated in favor of style sheets.
- Size - Used to define the size of the enclosed text, the size attribute has been deprecated in favor of style sheets.
- Start - The start attribute of list tags defines the starting number of the list. This attribute has been deprecated in favor of style sheets.
- Text - The text attribute defines the color of the text on the Web page. It has been deprecated in favor of style sheets.
- Type - The type attribute of list tags defines what type of list should be used. This attribute has been deprecated in favor of style sheets.
- Value - The value attribute of list tags sets the value of the list item. This attribute has been deprecated in favor of style sheets. This attribute is still applicable to <input> tags.
- Version - The version attribute defines the version of HTML used by the document. It has been deprecated in favor of DTDs.
- Vlink - The vlink attribute defines the color of a visited link on the Web page. It has been deprecated in favor of style sheets.
- Width - The width attribute defines the width in pixels of the element. It has been deprecated in favor of style sheets.

4.11 XHTML Forms

All forms within a website must be consistent and follow the strict accessibility guidelines so that users with disabilities can still fill out the form, the exact specification is beyond the scope of this document, but a sample form will be provided within the website templates as covered in the next section

4.12 Form Email Address / Password fields

All form email and password fields, there must be a confirmation box where the details can be re-entered. These details must be validated by Javascript to ensure they are the same.

4.13 Address Tag

The <address> tag defines the start of an address. You should use it to define addresses, signatures, or authorships of documents. Although addresses usually consist of several lines, line breaks within <address> ... </address> are not honoured and it is necessary to use
 tags to cause line breaks.

This is an important tag with regards to search results where local results are required.

5 G-Forces Templates

5.1 Template Location

When a new website is being created the default template provided within the Development folder in the base G-Forces Client Folder Template. The following should also be applied:

- Alt tags must be applied to:
  - Menu images (if appropriate)
  - Main content images
  - Main template logos and images
- Main menu items will include access keys and will have a logical “tabindex”
- Global JavaScript files will be included as a separate file – not added to each page, page specific functions at bottom

5.2 Folder Structure

There is now be a generic folder structure that must be followed for all PHP projects. Additional folders may added as necessary, redundant directories may be removed as necessary.

/classes/	Any G-Forces PHP classes
/classes/DataObjects/	DataObjects compiled here
/lib/	Any 3rd party classes, each within its own directory
/lib/pear/	All PEAR repository classes
/templates/	All template files
/templates/compiled/	Flexy compiled templates
/www/	Main web site root ie www.site.com points here
/www/css/	All CSS files
/www/js/	All JavaScript files
/www/edit/	Client login to back end ie www.site.com/edit/
/www/images/	All image files – may include extra sub folders
/www/images/cms_layup/	All backend layup images
/www/images/layup/	All layup images
/www/upload/	All uploaded files (images/pdf etc), may contain sub folders

5.3 Subversion Repositories

It is the responsibility of the template creator to create the relevant subversion repository and add all the information for templates, etc. into subversion. Once the repository has been created and the templates, etc. added to the repository, the development folders and files on Trinity must be removed and a text file created which provides the location of the subversion repository. Any previous versions, should be zipped to avoid any confusion between versions.

5.4 Robots.txt

All sites should have a robots.txt in the root of the site to control search engine indexing of certain areas, here is an example of what robots.txt should look like:

```
User-agent: *
Disallow: /css/
Disallow: /edit/
Disallow: /flash/
Disallow: /js/
```

6 Database

6.1 Database names

Databases should be named using all lowercase characters to match the master domain to which the database belongs. Periods and hyphens will need to be replaced with underscores, for example:

www_gforces_co_uk

6.2 Table names

Tables should be named using all lowercase characters, with underscores ('_') separating words. Table names should not with a digit. Table names must be pluralized where applicable. Periods and hyphens will need to be replaced with underscores, for example:

account_details

When a table is part of a distinct PHP Package, its tables should be prefixed with the package name in all lowercase and separated by an underscore. Package names may be shortened if necessary, but should be done so consistently. For example:

auth_user_permissions

6.3 Field Names

Fields should be named using all lowercase characters, with underscores ('_') separating words.

last_accessed

Primary key fields must always be named id. Foreign key fields should always be named by the name of the table they refer to (in singular excluding any package prefix) and appended with _id. For example:

Foreign Key linking to a table which is part of the 'auth_' package and called 'auth_user_permissions' would be:

user_permission_id

Also for the sake of consistency, tables should always be named in plural and field names always in the singular.

Bad example:
user_goups

Good Example:
user_group

6.4 Indexes/Unique Keys

Identify indexs and unique keys as part of the initial design process, as well as ongoing development as you develop new queries. If you are fetching data from a table based on a field or group of fields (in the where clause), ensure that there is an index setup for the field(s). This will mean that the database only has to read the rows which it needs to return rather than having to check every single record in the database. Also, if a field is going to be unique, then make it unique in the database structure. This will also increase speed as well as guaranteeing that a field with the same value will never be inserted more than once.

To guard against slow queries degrading performance on the shared database infrastructure, the most commonly performed SQL queries can be analysed with an SQL explain command. This will tell you which fields are being accessed from the query, and which index is being used to find the data.

~~6.5 Administration Users Table~~

~~All projects must have a table called admin_users which stores all of the authorised users who are able to administer the site, the table must follow this structure:~~

```
CREATE TABLE admin_users (
        id mediumint(6) unsigned NOT NULL auto_increment,
        admin_firstname varchar(100) default NULL,
        admin_surname varchar(100) default NULL,
        admin_email varchar(100) default NULL,
        admin_password varchar(100) default NULL,
        PRIMARY KEY  (id),
        KEY admin_email (admin_email)
) TYPE=MyISAM
```

~~Any internal or external testing/demostrations must utilise the correct admin_email and admin_password combination. Admin/test, admin/admin, etc. must never be used as a e-mail address/password combination.~~

6.6 E-mail Log Table

All projects must have a table called email_log which stores all of the e-mails generated by the site, the table must follow this structure:

```
CREATE TABLE email_logs (
        id mediumint(6) unsigned NOT NULL auto_increment,
        email_type varchar(50) default NULL,
        to_addresses varchar(250) default NULL,
        cc_addresses varchar(250) default NULL,
        bcc_addresses varchar(250) default NULL,
        email_message text,
        sent timestamp(14) NOT NULL,
        user_ipaddress varchar( 50) default NULL,
        PRIMARY KEY  (id)
) TYPE=MyISAM
```

Any entries that are older than 3 months old can be automatically deleted from the table.

6.7 Security

Database users should only be granted sufficient privileges to enable them to perform the tasks that they need to.

When connecting from web applications, where possible always connect to the database as the same user. This minimizes the chances of leaving a security hole against a particular user, and enables connection pooling techniques to be more readily taken advantage of to improve performance.

Plain text passwords should never be displayed in client side code.

7 ASP Code

7.1 Default Document

The default document must always be index.asp

7.2 Naming Conventions

There is not a naming convention for filenames, but the filename should point towards the function of the page, all lowercase letters should be used and _ if a space is required:

product.asp
database_functions.asp

7.2.1 Inbuilt Features

Usage of the inbuilt features of ASP (e.g. Functions, Keywords, Methods, etc.) should be utilised and capitalized as per the reference guide by Microsoft at http://msdn.microsoft.com/library/en-us/script56/html/vbscripttoc.asp

Some examples:

FormatDateTime(Date, vbShortDate)
MonthName(10, True)
Server.CreateObject("Scripting.FileSystemObject")

7.2.2 Custom Functions, Subroutines

Functions and subroutines should be named using the "studly caps" style (also referred to as "bumpy case" or "camel caps"). The initial letter of the name is lowercase, and each letter that starts a new "word" is capitalized.

Some examples:

Function connect()
Function getData()
Sub buildSomeWidget()

7.2.3 Variables

Variables must declared using the Dim statement, they should also be named using the "studly caps" style and Hungarian Notation i.e. the first three letters of the data type to the beginning of a variables name to distinguish the type of data that the variable contains. The table below demonstrates the usage of this naming convention and gives some examples of it in action:

Data Type	Prefix	Example Usage
Boolean	bln	blnActiveAccount
Date	dat	datToday
Decimal	dec	decMonthlySalary
Double	dbl	dblGrandTotal
Integer	int	intRows
Object	obj	objMail
String	str	strCompanyName

Names should be descriptive, but concise. We don't want huge sentences as our variable names, but typing an extra couple of characters is always better than wondering what exactly a certain variable is for.

7.2.4 Session Variables

Session variables should be avoided at all costs as they have implications on server efficiency, cookies should be used instead. The only acceptable use of session variables is to use the SessionID to track a user through the site to maintain a shopping cart, etc. Usage of the SessionID:

```
Session.SessionID
```

7.3 Control Structures

These include if, for, while, select case, etc. Here is an example if statement, since it is the most complicated of them:

```
If condition1 Or condition2 Then
        action1
Else If condition3 And condition4 Then
        action2
Else
        Defaultaction
End If
```

Multiple if statements should be moved to a Select Case statement if possible. Case's can have more than one condition, Select Case statements should follow the following format:

```
Select Case condition
        Case 1, 2
                action1
        Case 3
                action2
        Case Else
                Defaultaction
End Select
```

7.3.1 Inline conditionals

Inline conditionals should only be used to do very simple things. Preferably, they will only be used to do assignments, and not for function calls or anything complex at all. They can be harmful to readability if used incorrectly, so don't fall in love with saving typing by using them, example:

```
If intRowCount = 0 Then strVariable = "yes" Else strVariable = "No"
```

7.4 Database

7.4.1 Database Connections

All database connections must be opened and closed correctly to maximise efficiency and allow for proper connection pooling. For example:

Open database:

```
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.ConnectionString = ConnectionString
Conn.Open
```

Close database:
Conn.Close
Set Conn = Nothing

7.4.2 Database RecordSets & Arrays

RecordSets should returned to the application and placed into an array so that the RecordSet and Connection can be closed to maximise efficiency and allow for proper connection pooling. For example:

Open database as above
Set RS = Server.CreateObject("ADODB.RecordSet")
RS.Open SQL, Conn, adOpenForwardOnly, adLockReadOnly

If Not RS.EOF And Not RS.BOF Then
    arrRecordSet = RS.GetRows
End If
RS.Close
Set RS = Nothing
Close database as above

7.4.3 Database Constants

Database constants should be used at all times, by default, enumerated constants are not defined in VBScript and therefore you must include adovbs.inc in your pages to allow access to these constants. By default adovbs.inc should be installed at C:\Program Files\Common Files\System\ADO on your machine.

8 PHP Code

8.1 Naming Conventions

Use the PEAR naming standard which observes the following rules:

```
MyClass.php //  any file with a class definition
file.php    //  any file with procedural code
```

8.1.1 Classes

Classes should be given descriptive names. Avoid using abbreviations where possible. Class names should always begin with an uppercase letter. Each subsequent word in a class name should be capitalised and adjoining the previous word with no underscores. When a class is part of a distinct package of classes, they should all be prefixed by the common package name hierarchy. A package may in turn contain sub-packages. Each level of the hierarchy must be separated with a single underscore. Examples of good class names are:

```
HTTP_FileUploader
Authentication_User
```

The one-class-per-file convention should be used, unless it is considerably incontinent to do so, i.e., for performance reasons.

Examples of classes which are part of a package are as follows:

```
Foo/Bar/Baz.php        // filename
class Foo_Bar_Baz  {}   // classname
```

This convention is especially important for all files which are intended for reuse and entry in the shared repository. See the PEAR repository for many more examples.

The most convincing reason to use this file naming convention, which means that a class located in your include path like Foo/Bar/Baz.php is called Foo_Bar_Baz, is the ability to take advantage of PHP 5's __autoload magic method.

What this means is that if you instantiate the above class, and did not include the source file, it can be located and loaded automatically from your include path. Here's the code:

```
function __autoload($class) {
  $filename = strreplace('_', '/', $class) . '.php';
  @require_once $filename;
}
```

8.1.2 Functions, Methods

Functions and methods should be named using the "studly caps" style (also referred to as "bumpy case" or "camel caps"). Procedural functions should in addition have the package name as a prefix, to avoid name collisions between packages. The initial letter of the name (after the prefix) is lowercase, and each letter that starts a new "word" is capitalized. Some examples:

Methods:

```
connect()
getData()
buildSomeWidget()
```

Functions:
XML_RPC_serializeData()

8.1.3 Variables

Variables should also be named using the "studly caps" style (also referred to as "bumpy case" or "camel caps"). Names should be descriptive, but concise. We don't want huge sentences as our variable names, but typing an extra couple of characters is always better than wondering what exactly a certain variable is for.

The only situation where a one-character variable name is allowed is when it's the index for some looping construct. In this case, the index of the outer loop should always be $i. If there's a loop inside that loop, its index should be $j, followed by $k, and so on. If the loop is being indexed by some already-existing variable with a meaningful name, this guideline does not apply, example:

```
for ($i = 0; $i < $outerSize; $i++) {
    for ($j = 0; $j < $innerSize; $j++) {
        foo($i, $j);
    }
}
```

8.1.4 Private/Protected Methods and Variables (php4 ONLY)

Private class members in PHP (meaning class members that are intended to be used only from within the same class in which they are declared; PHP4 does not support truly-enforceable private namespaces) are proceeded by a single underscore. For example:

```
_sort()
_initTree()
$this->_status
```

8.1.5 Constants

Constants should always be all-uppercase, with underscores to seperate words. Prefix constant names with the uppercased name of the class/package they are used in. For example, the constants used by the DB:: package all begin with "DB_".

8.1.6 Global Variables

Global variables should be avoided at all costs, but if your package needs to define global variables, their name should start with a single underscore followed by the package name and another underscore. For example, the PEAR package uses a global variable called $_PEAR_destructor_object_list.

8.2 Control Structures

These include if, for, while, switch, etc. Here is an example if statement, since it is the most complicated of them:

```
if ( ( condition1 ) || ( condition2 ) ) {
   action1;
} else if ( ( condition3 ) && ( condition4 ) ) {
   action2;
} else {
   defaultaction;
}
```

Control statements should have one space between the control keyword and opening parenthesis, to distinguish them from function calls. They should then also have a space after the opening parenthesis and before the closing parenthesis. Parenthesis should be used around groups of conditions where appropriate if it makes the code easier to read.

You are strongly encouraged to always use curly braces even in situations where they are technically optional. Having them increases readability and decreases the likelihood of logic errors being introduced when new lines are added.

Switch statements should always define a default case when not doing so could lead to a potential security hole. Switch statements should follow the following format:

```
switch ( condition ) {
   case 1:
      action1;
   break;

   case 2:
      action2;
   break;

   default:
      defaultaction;
   break;
}
```

NEVER EVER USE INLINE SWITCH STATEMENTS. For what they save in page space they reduce, they lose a lot more in readability.

8.2.1 Inline conditionals

Inline conditionals should only be used to do very simple things. Preferably, they will only be used to do assignments, and not for function calls or anything complex at all. They can be harmful to readability if used incorrectly, so don't fall in love with saving typing by using them, examples:

Bad place to use them:

```
(($i < $size) && ($j > $size)) ? doStuff($foo) : doStuff($bar);
```

Ok place to use them:

```
$minSize = ( $i < $size ) ? $i : $size;
```

8.3 Function Calls

Functions should be called with no spaces between the function name, the opening parenthesis, and the first parameter; spaces between commas and each parameter, and no space between the last parameter, the closing parenthesis, and the semicolon. Here's an example:

```
$var = foo( $bar, $baz, $quux );
```

As displayed above, there should be one space on either side of an equals sign used to assign the return value of a function to a variable. In the case of a block of related assignments, more space may be inserted to promote readability:

```
$short         = foo( $bar );
$long_variable = foo( $baz );
```

8.4 Function Definitions

Function declarations follow the PEAR convention:

```
function fooFunction( $arg1, $arg2 = '' ) {
   if ( condition ) {
      statement;
   }
   return $value;
}
```

Arguments with default values go at the end of the argument list. Always attempt to return a meaningful value from a function if one is appropriate. Here is a slightly longer example:

```
function connect( &$dsn, $persistent = false ) {
   if ( is_array( $dsn ) ) {
      $dsninfo = &$dsn;
   } else {
      $dsninfo = DB::parseDSN( $dsn );
   }

   if ( !$dsninfo || !$dsninfo['phptype'] ) {
      return $this->raiseError();
   }

   return true;
}
```

8.5 PHP Code Tags

Always use <?php ?> to delimit PHP code, not the <? ?> shorthand.

8.6 Including Code

Anywhere you are unconditionally including a class file, use require_once(). Anywhere you are conditionally including a class file (for example, factory methods), use include_once(). Either of these will ensure that class files are included only once. They share the same file list, so you don't need to worry about mixing them - a file included with require_once() will not be included again by include_once().

When developing in php 5, __autoload should be utilised (as described above) to include files which are part of a package.

8.7 Comments

Inline documentation for classes should follow the PHPDoc convention, similar to Javadoc. More information about PHPDoc can be found here: http://www.phpdoc.org/

Non-documentation comments are strongly encouraged. A general rule of thumb is that if you look at a section of code and think "Wow, I don't want to try and describe that", you need to comment it before you forget how it works.

C style comments (/* */) and standard C++ comments (//) are both fine. Use of Perl/shell style comments (#) is discouraged.

8.8 General Guidelines

8.8.1 Quoting strings

There are two different ways to quote strings in PHP - either with single quotes or with double quotes. The main difference is that the parser does variable interpolation in double-quoted strings, but not in single quoted strings. Because of this, you should always use single quotes unless you specifically need variable interpolation to be done on that string. This way, we can save the parser the trouble of parsing a bunch of strings where no interpolation needs to be done.

Also, if you are using a string variable as part of a function call, you do not need to enclose that variable in quotes. Again, this will just make unnecessary work for the parser. Note, however, that nearly all of the escape sequences that exist for double-quoted strings will not work with single-quoted strings. Be careful, and feel free to break this guideline if it's making your code harder to read, examples:

Incorrect:

```
$str = "This is a really long string with no variables for the parser to find.";
do_stuff("$str");
```

Correct:

```
$str = 'This is a really long string with no variables for the parser to find.';
do_stuff( $str );
```

8.8.2 Associative array keys

In PHP, it's possible to use a literal string as a key to an associative array without quoting that string. We don't want to do this - the string should always be quoted to avoid confusion. Note that this is only when we're using a literal, not when we're using a variable, examples:

Incorrect:

```
$foo = $assoc_array[blah];
```

Correct:

```
$foo = $assoc_array['blah'];
```

8.8.3 Magic numbers

Don't use them. Use named constants for any literal value other than obvious special cases. Basically, it's OK to check if an array has 0 elements by using the literal 0. It's not OK to assign some special meaning to a number and then use it everywhere as a literal. This hurts readability AND maintainability. Included in this guideline is that we should be using the constants TRUE and FALSE in place of the literals 1 and 0 - even though they have the same values, it's more obvious what the actual logic is when you use the named constants.

8.8.4 Shortcut operators

The only shortcut operators that cause readability problems are the shortcut increment ($i++) and decrement ($j--) operators. These operators should not be used as part of an expression. They can, however, be used on their own line. Using them in expressions is just not worth the headaches when debugging, examples:

Incorrect:

```
$array[++$i] = $j;
$array[$i++] = $k;
```

Correct:

```
$i++;
$array[$i] = $j;

$array[$i] = $k;
$i++;
```

8.9 Header Comment Blocks

All source code files shall contain a "class-level" docblock immediately above each class. Below are examples of such docblocks.

```
<?php
/**
 * Short description for class
 *
 * Long description for class (if any)...
 *
 * PHP Version
 *
 *  @author     Original Author <author at example.com>
 *  @author     Another Author <another at example.com>
 *  @version    Release:  at package_version at
 *  @see        NetOther, Net_Sample::Net_Sample()
*/
?>
```

See Appendix for description of elements.

8.10 Templates

Tab index on forms to use tabindex="{tabIndex()}" to generate automatic tab numbering. You must ensure that you have the TabIndex function in the templates class if you are using a legacy version of this.

8.11 Error Handling

All errors should be caught and dealt with in a professional manner. There should never be a case where errors are output directly to the browser in a released product.

Details of errors should be emailed to whoever is responsible for the project along with all relevant information which will aid the analysis process. In the case of a critical error, the end user should be redirected to an error page which informs them that there has been a problem and that a technician has been automatically informed. Measures should be taken to insure only a limited amount of errors are emailed at any one period to prevent a flood of emails.

8.12 PHPDoc: Required Tags That Have Variable Content

8.12.1 Short Descriptions

Short descriptions must be provided for all docblocks. They should be a quick sentence, not the name of the item. Please read the Coding Standard's Sample File about how to write good descriptions.

8.12.2 PHP Versions

One of the following must go in the docblock:

```
 * PHP version 4
 * PHP version 5
```

* PHP versions 4 and 5

8.12.3 @author

There's no hard rule to determine when a new code contributor should be added to the list of authors for a given source file. In general, their changes should fall into the "substantial" category (meaning somewhere around 10% to 20% of code changes). Exceptions could be made for rewriting functions or contributing new logic.

Simple code reorganization or bug fixes would not justify the addition of a new individual to the list of authors.

8.12.4 Optional Tags

8.12.4.1 @see

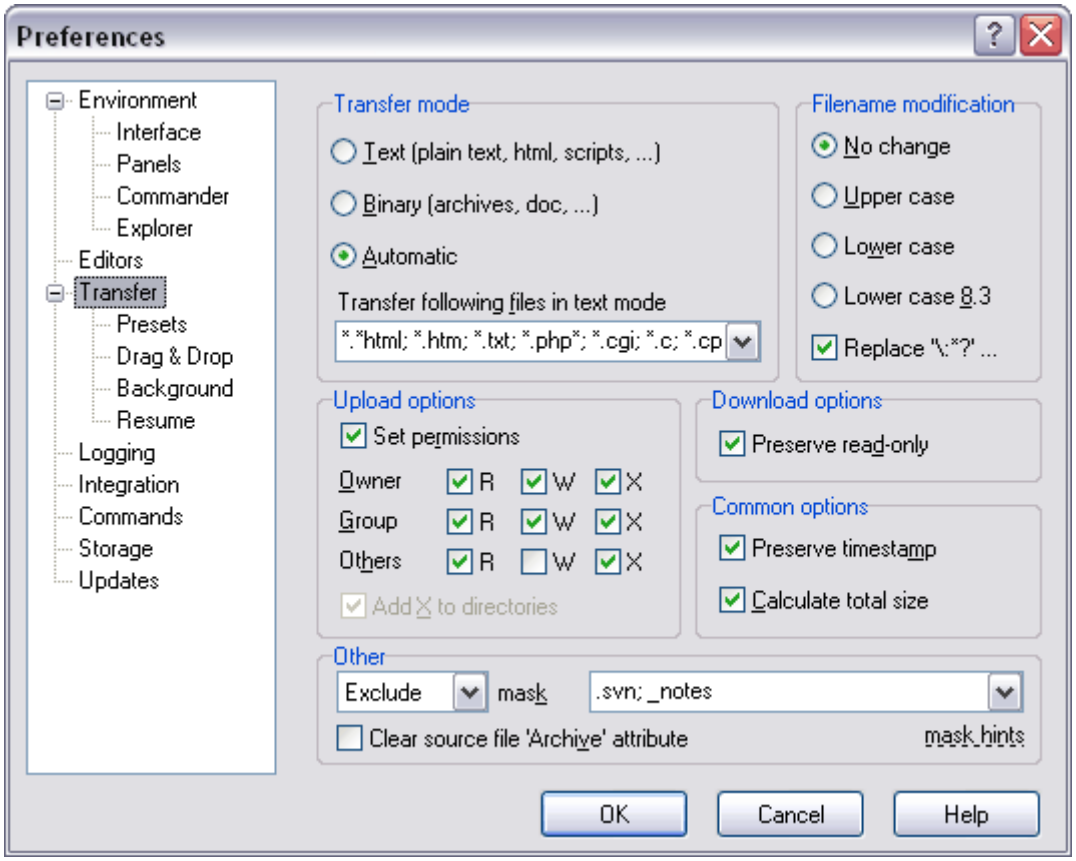
Add an @see tag when you want to refer users to other sections of the package's documentation. If you have multiple items, separate them with commas rather than adding multiple at see tags.

9 Miscellaneous

9.1 Uploads

All uploads to the live server environment must be through WinSCP to correctly allow for the secure communication needed for the upload to complete correctly. WinSCP is a freeware utility and the installer can be found at \\Trinity\Development\Programmes & Utilities.

A small amendment will also be required to the transfer preferences to allow other users to overwrite any files or folders you have uploaded, go to Preferences --> Transfer and make sure that the Upload options have the same settings as the screen shot below:



Also on this screen make sure that .svn is excluded from the upload to avoid all of the Subversion file information being uploaded to the server.

If you create a folder on the server, you will need to manually change the permissions on the folder to the same as above to allow other users to change the folder if necessary.

10 Javascripts

10.1 Prototype Image Cropper

10.2 Prototype Validation

For this you will need to include prototype.js and validation.js

```
<form method="post" id=" FormNameGoesHere " flexy:ignore>
        Normal form goes here.
</form>

<script type="text/javascript">
<!--

new Validation('FormNameGoesHere');

→
</script>
```

When the form is submitted the function will be automatically called and validated

Name	What it validates
required	This is a required field
validate-number	Please enter a valid number in this field.
validate-nonZeroNumber	Number must be greater than 0
validate-digits	Calidates true for 0 - 9
validate-alpha	Validates only letter a - z
validate-alphanum	Validates a – z + 0 - 9
validate-date	Validates a date mm/dd/yyyy
validate-require-email	Email is required and must be valid
validate-email	Email not required but must be valid if entered
validate-url	Validates the url to be http, https, ftp
validate-date-gb	Validates a date dd/mm/yyyy
validate-currency-dollar	Validates a currency including the $ sign and , and .
validate-one-required	Atleast one input under the same name is required.
validate-ukpost	Validates a UK postcode
validate-ukphone	+44 then standard number including spaces

10.3 Lightbox & Lightbox2

DO NOT USE UNLESS THERE ARE EXCEPTIONAL CIRCUMSTANCES. SEE A SENIOR DEVELOPER BEFORE USING THIS.

10.4 Prototype In General

For more information read the Prototype API pdf book. The Prototype Cheat sheet.
Or online at http://www.prototypejs.org there is a full tutorial plus reference API.

11 Search Engine Optimisation

11.1.1 Essentials:

- Ascertain Keywords or phrases used within the site Title
- Page Content, position and emphasis on keywords/key phrases

- Good Meta Tags – Title, Description & Keywords
- List site on Open Directory (www.dmoz.org) with keyword relevant description
- Number of inbound hyperlinks from other relevant and quality websites

11.1.2 Constructing Meta Tags:

11.1.2.1 Title:

- 50 to 80 characters long including spaces
- Include 1 to 2 of the most important keyword phrases
- Place keyword phrases at beginning of title
- Each page should have its own title and keyword phrases

E.g. 'Timber Garages'

11.1.2.2 Description:

- Should match the title

11.1.2.3 Keywords:

- Key phrases improve odds of success – 1 to 2 word phrases
- Make sure target keywords appear in title
- Make sure target keywords appear high up on the web page
- If possible use the target keywords in the headline as well as in the first few lines of the first paragraph
- Keyword Meta no longer than 1000 characters – 10 keywords/ key phrases (25 max)
- Only use tags that appears in the title, description and other Tags + Content
- Make sure to include plurals
- Do not repeat a keyword more than 5 times in combination phrases

11.1.2.4 ALT Text:

- Add ALT text over images
- Use phrases that appears in the title, description and other Tags + Content
- Only use a couple of phrases for each image ALT tag
- Make the ALT tag readable

11.1.3 Hyperlink Optimisation:

- Place keyword phrases in links themselves and in t he text immediately before or after the link

11.1.4 Content:

- Ensure at least 200 words of copy per page for best results
- Text should include keyword phrases
- Must be readable and not simply lists of keywords
- Ensure content and Meta Tags reflect each other
- Ensure bold titles at top of page

11.2 Submission Process:

11.2.1 Stage 1:

- Select Key phrases – fewer the better
- Check via overture account to measure traffic levels – adjust where necessary
- Develop Meta Tags – Titles, Description and Keywords
- Adjust home page content to include Key phrase references

11.2.2 Stage 2:

- Add to Open Directory with keyword relevant description (www.dmoz.org)
- Add to Google – www.google.co.uk/addurl
- Add to Yahoo network: http://submit.search.yahoo.com/free/request (need to have a Yahoo ID)
  Yahoo! ID: gforcesuk
  Password: lemons