PHP Naming Conventions

Classes

HUBzero Library

HUBzero Core Library uses the PSR-0 class naming convention whereby the names of the classes directly map to the directories in which they are stored. The root level directory ofHUBzero's standard library is the "Hubzero/" directory. All HUBzero core library classes are stored hierarchically under these root directories.

Class names may only contain alphanumeric characters. Numbers are permitted in class names but are discouraged in most cases. Underscores are only permitted in place of the path separator; the filename "Hubzero/User/Profile.php" must map to the class name "HubzeroUserProfile".

If a class name is comprised of more than one word, the first letter of each new word must be capitalized. Successive capitalized letters are not allowed, e.g. a class "HubzeroPDF" is not allowed while "HubzeroPdf" is acceptable.

Note: Code that must be deployed alongside Hubzero and Joomla libraries but is not part of the standard or extras libraries (e.g. application code or libraries that are not distributed by Hubzero) must never start with "Hubzero".

Extensions

Classes should be given descriptive names. Avoid using abbreviations where possible. Class names should always begin with an uppercase letter and be written in CamelCase even if using traditionally uppercase acronyms (such as XML, HTML).

Namespaced

While namespacing an extension is not required, it is encouraged.

Controllers

For components, such as the Blog in the Administrator, the convention is Components[Component]Controllers[Name].

```
namespace ComponentsBlogControllers;

use HubzeroComponentAdminController;

class Entries extends AdminController
{
    // Methods
```

```
}
```

Models

The naming convention is Components[Component]Models[Name].

```
namespace ComponentsBlogModels;

use HubzeroBaseModel;

class Entry extends Model
{
    // Methods
}
```

Plugins

Currently, plugin naming must follow the pseudo-namespace conventions.

non/pseudo-Namespaced

These conventions define a pseudo-namespace mechanism for extensions in the framework. Third-party developers are to avoid beginning names with ‘Hubzero’ as it is reserved. It is advisable for developers to name classes with their own unique prefix.

Controllers

For single controller components, the naming convention is [Component]Controller.

```
class ContentController extends HubzeroComponentSiteController
{
    // Methods
}
```

For a multi-controller components, such as the Blog in the Administrator, the convention is [Component]Controller[Name].

```
class BlogControllerEntries extends HubzeroComponentAdminController
```

```
{
    // Methods
}
```

Models

The naming convention is [Component]Model[Name].

```
class BlogModelEntry extends HubzeroBaseModel
{
    // Methods
}
```

Plugins

The naming convention is plg[Folder][Element]

```
class plgContentPagebreak extends HubzeroPluginPlugin
{
    // Methods
}
```

Filenames

Only alphanumeric characters, underscores, and the dash character (”-”) are permitted. Spaces are strictly prohibited.

Any file that contains PHP code should end with the extension “.php”. The following examples show acceptable filenames:

```
Hubzero/Session/Helper.php
```

```
Hubzero/View/Helper/Html.php
```

Controllers

For single controller components, the naming convention of

Components[Component][Client]Controller will map to a file name of controller.php and be located in the component folder.

```
com_content
.. /site
.. .. controller.php
```

For a multi-controller components, the convention of Components[Component][Client]Controllers[Name] will map to files located in a /controllers folder under the component folder. The file names will reflect the name of the controller.

```
com_blog
.. /site
.. .. /controllers
.. .. .. entries.php
.. .. .. media.php
```

Models

The naming convention of [Component]Model[Name] will map to a similar file structure. The files will be located in a /models folder under the component folder. The file names will reflect the name of the model.

```
com_blog
.. /models
.. .. entry.php
.. .. comment.php
```

Layouts

Components may support different Layouts to render the data supplied by a View and its Models. A Layout file usually contains markup and some PHP code for display logic only: no functions, no classes.

A Layout consists of at least one .php file and an equally named .xml manifest file located in the /tmpl/ folder of a View, both reflect the internal name of the Layout. The standard Layout is called “display”.

```
com_blog
.. /site
.. .. /views
.. .. .. /entries
.. .. .. .. /tmpl
.. .. .. .. .. display.php
.. .. .. .. .. display.xml
.. .. .. .. .. edit.php
.. .. .. .. .. edit.xml
.. .. .. .. .. entry.php
.. .. .. .. .. entry.xml
```

Functions and Methods

Function names may only contain alphanumeric characters. Underscores are not permitted except as a prefix to indicate protected or private methods. Numbers are permitted in function names but are discouraged in most cases.

Function names must always start with a lowercase letter. When a function name consists of more than one word, the first letter of each new word must be capitalized. This is commonly called “camelCase” formatting.

Verbosity is generally encouraged. Function names should be as verbose as is practical to fully describe their purpose and behavior.

These are examples of acceptable names for functions:

```
filterInput()

getElementById()

widgetFactory()

_myPrivateMethod()
```

For object-oriented programming, accessors for instance or static variables should always be prefixed with “get” or “set”. In implementing design patterns, such as the singleton or factory patterns, the name of the method should contain the pattern name where practical to more thoroughly describe behavior.

For methods on objects that are declared with the “private” or “protected” modifier, the first

character of the method name must be an underscore. This is the only acceptable application of an underscore in a method name. Methods declared “public” should never contain an underscore.

Functions in the global scope (a.k.a “floating functions”) are permitted but discouraged in most cases. Consider wrapping these functions in a static class.

Variables

Variable names may only contain alphanumeric characters. Underscores and numbers are permitted in variable names but are discouraged in most cases.

For instance variables that are declared with the “private” or “protected” modifier, the first character of the variable name must be a single underscore. Member variables declared “public” should never start with an underscore.

As with function names (see above) variable names must always start with a lowercase letter and follow the “camelCaps” capitalization convention.

Verbosity is generally encouraged. Variables should always be as verbose as practical to describe the data that the developer intends to store in them. Terse variable names such as “$i” and “$n” are discouraged for all but the smallest loop contexts. If a loop contains more than 20 lines of code, the index variables should have more descriptive names.

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.

```
namespace HubzeroBase;

class Example
{
    private $_status = null;

    protected $_fieldName = null;

    protected function _sortNames()
    {
        $someNames = array();
    }
}
```

Constants

Constants may contain both alphanumeric characters and underscores. Numbers are permitted in constant names.

All letters used in a constant name must be capitalized, while all words in a constant name must be separated by underscore characters.

For example, EMBED_SUPPRESS_EMBED_EXCEPTION is permitted but EMBED _SUPPRESSEMBEDEXCEPTION is not.

Prefix constant names with the uppercase name of the class/package they are used in. For example, the constants used by the JError class all begin with “JERROR_”.

Constants must be defined as class members with the “const” modifier. Defining constants in the global scope with the “define” function is permitted but strongly discouraged.