Structure

Naming Conventions

The model, view and controller files use classes from the framework, HubzeroBaseModel, HubzeroComponentView and HubzeroComponentSiteController, respectively. Each class is then extended with a new class specific to the component.

Administrative controllers extend HubzeroComponentAdminController which in turn extends HubzeroComponentSiteController and adds a few extra methods frequently used throughout the administrative portion of the site.

All components must be under the Components namespace and follow PSR-0 naming scheme with one exception: files and folders may be lowercase even when their class names are not.

Directories & Files

Components follow the Model-View-Controller (MVC) design pattern. This pattern separates the data gathering (Model), presentation (View) and user interaction (Controller) activities of a module. Such separation allows for expanding or revising properties and methods of one section without requiring additional changes to the other sections.

In its barest state, no database entry or other setup is required to "install" a component. Simply placing the component into the /components directory will make it available for use. However, if a component requires the installation of database tables or configuration (detailed in the config.xml file), then an administrator must install the component using one of the installation options in the administrative back-end.

Note: Components not installed via one of the installation options or without a database entry in the #__extensions table will not appear in the administrative list of available components.

To illustrate the typical component directory structures and files:

```
/app
.. /components
.. .. /com_example
.. .. .. /admin
.. .. .. /api
.. .. .. /helpers
.. .. .. /migrations
.. .. .. /models
.. .. .. /site
.. .. .. .. /assets
.. .. .. .. .. /css
```

```
.. .. .. .. .. /js
.. .. .. .. .. /img
.. .. .. .. /controllers
.. .. .. .. .. example.php
.. .. .. .. /views
.. .. .. .. .. /example
.. .. .. .. .. .. /tmpl
.. .. .. .. .. .. .. display.php
.. .. .. .. .. .. .. display.xml
.. .. .. .. example.php
.. .. .. .. router.php
.. .. .. /tests
```

Files are contained within directories titled "com_example". Some directories and files are optional but, for this example, we've included a more common setup.

All client-specific files and sub-directories are split between the respective client directories, such as admin and site. Since controllers and views are specific to a client, they reside within those client directories. Shared files, typically models and helpers, are within directories at the same level as the client folders.

Directory & File Explanation

/com_{componentname}/{client}/{componentname}.php
This is the component's entry point for the admin and site clients. API and Cli (console) are special cases and don't require this file.
/com_{componentname}/{client}/views

This folder holds the different views for the component.

/com_{componentname}/views/{viewname}
This folder holds the files for the view {ViewName}.
/com_{componentname}/views/{viewname}/tmpl

This folder holds the template files for the view {ViewName}.

/site/views/{viewname}/tmpl/default.php
This is the default template for the view {ViewName}.
/com_{componentname}/migrations

Database migrations for this component. The first or oldest migration will typically be the initial installation, creating any necessary database tables or seeding required data. Subsequent migrations will handle any schema or global data changes needed as the code evolves.

/com_{componentname}/models

This folder holds additional models, if needed by the application.

/com_{componentname}/models/{modelname}.php

This file holds the model class {ComponentName}Model{ModelName}. This class must extend the base class "HubzeroBaseModel". Note that the view named {ViewName} will by default load a model called {ViewName} if it exists. Most models are named after the view they are intended to be used with.

/com_{componentname}/{client}/controllers

This folder holds additional controllers, if needed by the application.

/com_{componentname}/{client}/controllers/{controllername}.php
This file holds the controller class {ComponentName}Controller{ControllerName}. This class must extend the base class HubzeroComponentsSiteController.

/com_{componentname}/tests

Unit tests for your code. These tests can be run through the command line **muse** application that comes with a HUBzero install.

Entry Point

The CMS is always accessed through a single point of entry: index.php for the Site Application or administrator/index.php for the Administrator Application. The application will then load the required component, based on the value of 'option' in the URL or in the POST data. For our component, the URL would be:

```
For search engine friendly URLs:
/hello

For non-SEF URLs:
/index.php?option=com_hello
```

This will load our main file, which can be seen as the single point of entry for our component: components/com_hello/site/hello.php.

Implementation

```
<?php
// Define the namespace
// Components{ComponentName}{ClientName};
namespace ComponentsHelloSite;

// Get the requested controller
$controllerName = Request::getCmd('controller', Request::getCmd('view'
, 'one'));

// Ensure the controller exists
if (!file_exists(__DIR__ . DS . 'controllers' . DS . $controllerName .
 '.php'))
{
 App::abort(404, Lang::txt('Controller not found'));
}
require_once(__DIR__ . DS . 'controllers' . DS . $controllerName . '.p
hp');
$controllerName = __NAMESPACE__ . '\Controllers\' . ucfirst(strtolower
($controllerName));

// Instantiate controller
$controller = new $controllerName();
// Execute whatever task(s)
$controller->execute();
```

The first statement is defining the namespace. All component namespaces **must** be under the Components namespace.

__DIR__ is a pre-defined PHP constant that evaluates to the absolute path to the current directory, in our case /webroot/app/components/com_hello/site.

DS is and alias of PHP's DIRECTORY_SEPARATOR constant and produces the directory separator of your system: either "/" or "". This is automatically set by the framework so the developer doesn't have to worry about developing different versions for different server OSes.

Most modern systems can now handle paths designated with a forward slash "/" so use of the DS constant is no longer strictly required or necessary.

First we look for a requested controller name. There is a default set in case none has been passed or if the requested controller is not found. With the controller name, we build the class name for the controller following the standard namespaced camel-cased pattern of Components{Component name}{Client name}Controllers{Controller name}

After the controller is created, we instruct the controller to execute the task, as defined in the URL: index.php?option=com_hello&task=sometask. If no task is set, the default task 'display' will be assumed. When display is used, the 'view' variable will decide what will be displayed. Other common tasks are save, edit, new...

The main entry point (hello.php) essentially passes control to the controller, which handles performing the task that was specified in the request.

Note that we don't use a closing PHP tag in this file: ?>. The reason for this is that we will not have any unwanted whitespace in the output code. This is default practice and will be used for all php-only files. Please see the coding Style Guide for further details.