# Introduction

## Getting Started

As a developer you are tasked with altering or extending the functionality of a HUBzero install or one of its extensions. You will need to be proficient in PHP and have some familiarity with such things as JavaScript or CSS. If you are new to HUBzero, this reference should help guide you through the creation of extensions such as modules and widgets (more on those later). Thankfully, the requirements for getting started creating HUBzero extensions are minimal: knowledge of programming in PHP and a good text editor. While those are the only *requirements* we do, however, recommend you have working knowledge of the following:

- HTML
- Cascading StyleSheets (CSS)
- JavaScript (familiarity with the jQuery framework is a plus)
- XML
- Model-View-Controller (MVC) design pattern
- Object-Oriented Programming

# Upgrade Guide

## Directory Structure & Files

Most notable about the 2.0.0 release will be the new directory structure and reorganization of the various files and extensions comprising the CMS.

Files are essentially divided between two primary directories: app and core.

```
/app
/core
index.php
```

The app directory is where everything concerning a specific hub lives. That is, it's the home to all the logs, cache data, uploads, and extensions unique to a specific instance of a hub.

## Constants

| Joomla | Hubzero |
| --- | --- |
| JPATH_ROOT | PATH_ROOT |
| JPATH_BASE | PATH_ROOT |
| JPATH_SITE | PATH_ROOT |
| JPATH_ADMINISTRATOR | PATH_ROOT<br>No files or code should be placed into or read from the administrator directory and it is slated for deletion in a future version. |
| JPATH_COMPONENT | Component::path($option) |
| n/a | PATH_APP<br>Points to ROOT/app where all hub-specific data resides. |
| n/a | PATH_CORE<br>Points to ROOT/core where the framework and core extensions live. |
| _JEXEC | _HZEXEC_ |

It is highly recommended, when including files within the same extension (component, module, plugin), to use the __DIR__ and __FILE__ PHP constants and relative paths.

```
<?php
// This file is example.php, located in:
// ROOT/app/components/com_example/admin
```

```
// dirname(__DIR__) moves up one directory
// ROOT/app/components/com_example/models
require_once(dirname(__DIR__) . DS . 'models' . DS . 'foo.php');

// ROOT/app/components/com_example/admin/controllers
require_once(__DIR__ . DS . 'controllers' . DS . 'example.php');
```

## Common Classes

To make upgrading an extension a little easier, a number of Joomla classes (and their methods) have equivalent classes in the new framework.

**JRoute**

| | |
|---|---|
| JRoute::_(); | Route::url(); |

**JText**

The JText class, used for translating language keys, was replaced by the Lang facade. Along with this, the _() and sprintf() methods were merged to allow for a single call to Lang::txt() with a variable number of arguments. If more than one argument is passed to the txt() method, the translator will attempt to perform variable replacement in the translated string.

```
// Language file
COM_EXAMPLE_HELLO="Hello!"
COM_EXAMPLE_HELLO_NAME="Hello, %s!"

...

// PHP

// Outputs 'Hello!'
echo Lang::txt('COM_EXAMPLE_HELLO');

// Outputs 'Hello, HUBzero!'
echo Lang::txt('COM_EXAMPLE_HELLO_NAME', 'HUBzero');
```

| | |
|---|---|
| JText::_(); | Lang::txt(); |
| JText::sprintf(); | Lang::txt(); |
| JText::plural(); | Lang::txts(); |
| JText::alt(); | Lang::alt(); |

**JRequest**

To make transitioning easier, all public JRequest methods have been preserved on the global request object, which can be accessed through the application container or the Request facade.

```
// Via the application container
$request = App::get('request');
$foo = $request->getVar('foo');

// Via the facade
$foo = Request::getVar('foo');
```

In the majority of cases, this means simply dropping the 'J' from JRequest will be sufficient for upgrading an extension's code.

| JRequest::* | Request::* |
| --- | --- |

**JToolbarHelper**

Perhaps one of the easier conversions; Simply replace instances of JToolbarHelper with the Toolbar facade. Method names and the arguments passed to them stay the same.

```
// Joomla
JToolbarHelper::publishList();
JToolbarHelper::unpublishList();

// Hubzero
Toolbar::publishList();
Toolbar::unpublishList();
```

**JSubMenuHelper**

As with JToolbarHelper above, only the class name need be updated. All primary method names stay the same.

```
// Joomla
JSubMenuHelper::addEntry(
 JText::_('COM_COLLECTIONS_POSTS'),
 'index.php?option=com_collections&controller=posts',
 $controllerName == 'posts'
);
```

```
// Hubzero
Submenu::addEntry(
 Lang::txt('COM_COLLECTIONS_POSTS'),
 Route::url('index.php?option=com_collections&controller=posts'),
 $controllerName == 'posts'
);
```

**JHtml**

Unlike may of the other classes mentioned above, the class, method, and arguments changed for the replacement of Html. Easily enough, the "J" can simply be dropped to have a class name of just Html. The method name and first argument passed to said method is a little more complicated but follows a strict pattern. For Html, all arguments were passed to a method of _(), the first argument being a dot-notation combination of sub library and the function to call within it.

```
echo JHTML::_('grid.sort', 'COM_COLLECTIONS_COL_TITLE', 'title', @$thi
s->filters['sort_Dir'], @$this->filters['sort']);
```

For the Html class, the method is now the name of the sub-library and the first argument passed is the name of the function to call.

```
echo Html::grid('sort', 'COM_COLLECTIONS_COL_TITLE', 'title', @$this->
filters['sort_Dir'], @$this->filters['sort']);
```

Examples:

```
// Joomla
JHtml::_('behavior.framework');

// Hubzero
Html::behavior('framework');
```

## Factory Objects

The following is a list of conversions for objects typically acquired from Joomla's JFactory. In most cases, the objects or their equivalents are available for retrieval from the global App. A number of the objects also have associated Facades for quicker access. In the examples below **method()** is variable and implies that the method formerly called on the Joomla object can be called statically on the facade.

Example 1:

```
// Joomla
$user = JFactory::getUser();
echo $user->get('name');

// Hubzero
echo User::get('name');
```

Example 2:

```
// Joomla
$doc = JFactory::getDocument();
$doc->addStylesheet('/some/file.css');

// Hubzero
Document::addStylesheet('/some/file.css');
```

| Joomla | Hubzero | Hubzero Facade |
|---|---|---|
| JFactory::getDbo(); | App::get('db'); | n/a |
| JFactory::getUser(); | User::getRoot(); | User::**method()**; |
| JFactory::getSession(); | App::get('session'); | Session::**method()**; |
| JFactory::getDocument(); | App::get('document'); | Document::**method()**; |
| JFactory::getConfig(); | App::get('config'); | Config::**method()**; |
| JFactory::getLanguage(); | App::get('lang'); | Lang::**method()**; |
| JFactory::getCache(); | App::get('cache.store'); | Cache::**method()**; |
| JFactory::getLogger(); | App::get('log.debug'); | Log::**method()**; |

## Dates

Along with a replacement class for Joomla's JDate, the CMS includes a global Date class to make handling and formatting of dates a little easier.

### Now

The Date class will always return an instance of HubzeroUtilityDate. If no specific time or timestamp is specified, it will default to 'now'.

```
// Output the current timestamp (UTC) in the database's format. ex: "2
015-04-03 12:23:56"
echo Date::toSql();

// Output the current timestamp (UTC) in Unix format.
echo Date::toUnix();

// Output the current timestamp (UTC) year. ex: "2015"
echo Date::format('Y');

// Output the current timestamp adjusted to the timezone of the hub. F
or example, if the UTC time is "12:23 pm" and the hub's set timezone i
s Eastern Standard Time (EST), the time outoutted will be "08:23 am"
echo Date::toLocal('g:i a');
```

### Specified date

A specific timestamp and timezone can be passed to the of method. If no timezone is provided, the timezone will default to UTC.

```
// Output the current timestamp (UTC) year. ex: "2013"
echo Date::of('2013-08-12 17:01:34')->format('Y');

// Output the current timestamp adjusted to the timezone of the hub. e
x: "1:01 pm"
echo Date::of('2013-08-12 17:01:34')->toLocal('g:i a');
```

## Users

The global user object, retrieved from JFactory::getUser() can now be accessed anywhere within the CMS from the User facade. Any method, other than getInstance(), statically called on User will be acted upon the current, global user. This is the same as calling JFactory::getUser()->method().

```
// Joomla
echo JFactory::getUser()->get('name');

// Hubzero
echo User::get('name');
```

The getRoot() method can be used to retrieve the underlying object (of the facade) and assigned to a variable as needed.

```
// Joomla
$user = JFactory::getUser();

// Hubzero
$user = User::getRoot();
```

Obtaining instances of new users can be achieved by calling getInstance($id_or_username) on the User facade in the same manner as calling JUser::getInstance($id_or_username) or JFactory::getUser($id_or_username).

```
// Joomla
$user = JFactory::getUser(1234);
// ... or ...
$user = JUser::getInstance(1234);

// Hubzero
$user = User::getInstance(1234);
```

# Contribution Guide

## Bug Fixes & Patches

The HUBzero Foundation accepts bug fixes and core patches. Submissions are reviewed by Foundation technical staff and, if accepted, they will become part of the HUBzero core distribution. When contributions are offered to the HUBzero Foundation, the copyright for the software must be reassigned to the HUBzero Foundation so that the changes can be managed as an indistinguishable part of the core distribution. The HUBzero Foundation reserves the right to reject submissions for any reason.

## Adherence to Standards

The HUBzero Foundation has established coding conventions and guidelines for developing HUBzero components. All submissions to the core must adhere to the guidelines, coding patterns, and styles laid forth in the documentation. Contributions are reviewed by the Foundation team before being incorporated into the core distribution and those that do not meet the standards may be tweaked or rewritten as needed, or may be accepted pending required changes. Any contribution known to contain security vulnerabilities may be rejected entirely.

## Hosted Hubs

All code extensions and alterations must adhere to the aforementioned coding conventions and guidelines for hubs hosted and maintained by HUBzero. Third-party extensions are allowable but must first be reviewed and accepted to ensure compatibility, functionality, and security concerns are addressed.

## Guidelines

- Be sure that the code follows the development styles and conventions
- Make sure it works.
- If it requires internationalization, documentation, or help to be written, be sure you've done that before submission.
- Variables, subroutines, and comments in the code should be made in English. While the talents of the worldwide community is greatly appreciated, we cannot support code that we cannot read.

## Subjective Requirements

Below are some subjective requirements that should be taken into consideration.

1. **Is this feature useful?**
   Consider whether this feature is useful to the current or future users. If there are relatively few people for who this is useful: what other reasons are there to include this? If this feature is useful only to some, can it be made in a way that it can be switched off

or easily removed, so that it's out of the way of the average user.

2. **Is this feature usable by the target audience?**
   Consider who is the target audience? Is it user-friendly for regular users, does a content manager in an avarage business environment have the skills to use it? Or is this directed to system administrators?
3. **What are the maintenance costs of it?**
   Strongly consider the time and effort spent in maintaining it.
4. **Does this fit in the direction HUBzero is going?**
   This may be something you want to discuss with other hub owners and users, but you can also submit inquiries to the HUBzero team.
5. **Is there already a similar feature?**
   If so: could this similar feature be adapted to suit you needs? Or does your contribution have all the features of the existing feature? And if so, is there a way to upgrade?