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 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 Points to ROOT/app where all hub-specific data resides.
n/a	PATH_CORE 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();	App::get('user');	User::**method()**;
JUser::getInstance();	User::getInstance();	
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')->logger('{log name}');	Log::**method()**; Note: The facade only interacts with the debug log. Use App::get('log'); to interact with any other log.

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

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

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

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

Overview

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 on the forum 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?

Code Guidelines

Hubzero maintains a set of guidelines and rules for code contributions. These are aimed at

creating a consistent, readable, security-minded, and maintainable codebase.

HUBzero Git Flow Slides

Standards and Code Styling

All code contributions must adhere to the style and conventions outlined in the following: Developer Conventions

PHP CodeSniffer

You can check your code using PHP CodeSniffer. This should be available in most repositories.

On the Amazon Instance or CentOS 6.5 installations, you may install this using sudo yum install phpcs Alternatively, you may install this from source: Code Sniffer - GitHUB

Once installed on your system you can clone the HUBzero Standards Repository from Hubzero Standards. It’s recommended to install the standards outside of the hub installation directory.

git clone -b hubzero-cms-2.1 https://github.com/hubzero/standards /home/<username>/standards

To run PHPCS, you would run an incantation similar to:

phpcs -np --standard=/home/<username>/standards/Php/ruleset.xml --ignore=*/test/*,*/assets/*,*/views/*,*/tmpl/*./path/to/code

Where <username> is in your Linux username (centos, more than likely if you're on an AWS instance).

Due to the size of the hubzero-cms codebase, it is recommended that you only run the check against files you've modified. Each file can be specified at the end of the command, replacing (./path/to/code), separated by spaces.

Suggested Workflow

Hubzero Web Developers have a couple of workflows which have developed organically over the years. In the future, we would like to use tooling to automate the development environment. Due to the complex configuration of all the hub software, starting from the VMware Image available for download.

A couple of developers have gotten together to put together a CMS-only environment using Vagrant: CMS-only Vagrant Environment. Instructions for using this environment are available in the repository's README file.

Configuration

The configuration necessary for web development is outlined below. There may be some variables such as login username, file locations, and users depending on the operating system. A person familiar with a LAMP environment should be able to ascertain these variables with relative ease. In all cases configuration secrets, such as MySQL root user password and admin user password are located in /etc/hubzero.secrets. The Hubzero command-line utility, hzcms, will create this file upon installation of the HUB.

External Contribution Process

Advanced Workflows

Unless you are confident in your Linux system administration skills, the Hub can be built via

Debian and RedHat packages. In the future, we anticipate moving to support RedHat / CentOS over Debian. In our effort to increase our presence on cloud service providers, RedHat / CentOS has been chosen to be implemented first.

Development Workflow on an AWS CentOS Image

Another approach, which is used internally at HUBzero, is to use the [Amazon Marketplace Image](#).

1. Create a GitHub account and Fork the hubzero cms repository
2. ssh centos@hostname.aws.hubzero.org
    1. Log into the development environment.
3. cd /var/www
    1. Go to the web root directory.
4. cv hub hub.bak
    1. Backup the existing code which will be necessary to preserve the configuration files.
5. git clone https://github.com/gitusername/hubzero-cms.git hub
    1. Clone the development repository, naming it “hub” to preserve the preconfigured location that the web server (Apache) expects.
6. git remote add upstream https://github.com/hubzero-cms
    1. This adds the HUBzero repository as your upstream remote. This allows you to pull down the latest history.
7. cp hub.bak/app hub/
    1. Copy the configuration, contained within the app directory, to the new installation.
8. cd hub/core
    1. a. Go to the core directory.
9. php composer install
    1. Pull in external dependencies (including the HUBzero Framework).
10. cd . . /
    1. Go back to the /var/www/hub directory.
11. php muse migration -i -f
    1. Run database migrations.
12. cd /var/www
    1. Go to the /var/www/ directory
13. chown apache:apahce hub - R
    1. Change the owner and group of the hub directory to something that the web server agrees with.
14. chmod 775 hub - R
    1. Change the file mode back to something permissible for development purposes. In production environments this is a little more locked down.
15. git checkout <branch>
    1. At the time of writing: 2.1.0 is the stable branch.
        1. This branch really should have been called 2.1 and is treated as such. Each minor release is branched from a point in time on 2.1.0.

    2. master is the unstable branch.
16. git fetch --all
    1. Pull the latest code.
17. git pull --rebase upstream/<branch>
    1. Merges the latest changes from upstream into your local branch.
18. git checkout -b <feature-branch>
    1. This creates a feature branch. This is something completely arbitrary that means something useful to you as a developer.
    2. This will create a branch from whatever branch you select in step 14 and will have the latest history from steps 15 and 16.
19. <make some valid changes to the code>
    1. Fix a bug, for instance.
20. git add <file>
    1. Add the file(s) to the commit.
21. git commit -m "Descriptive message"
    1. Form the commit and add a descriptive message meeting the standardized message formatting requirements.
22. git push origin <feature-branch>
    1. Push your changes to your repository.
23. Open Pull Request on GitHub
    1. This starts the code review process. Another developer must approve the code submission(s) before they enter the mainline.
    2. It is highly recommended that you run PHPCS against your code before opening pull request. Styling errors may lead to a rejection of your code.
24. For further contributions, repeat steps 15 - 23

General Tips

- Explore your dev environment. It’s yours! You should have your own personal development environment. Use it to explore the Hubzero platform as a whole. Most of the Hubzero code is written in PHP or Python, which can be read via a text editor.
- Don’t be afraid to “trash” your development environment. Development environments should be messy. They’re your workspace to make, break, and create cool stuff. In VMware and other virtual machine applications, creating snapshots is a convenient way to return the machine to a working state.
- Ask questions. Feel free to ask Hubzero Developers questions through support tickets or our Contributor’s channel at https://hubzerocontributors.slack.com.
- Read the documentation. We try to do our best to keep the essentials in our documentation up-to-date. If something doesn’t seem to be working for you, let us know! We’ll help you out.
- Use Muse. Muse is the CMS’s command-line interface. There is a lot of functionality it brings to the table.
- Submit Bug Fixes. Come across an issue in core? Submit a ticket or contribute a fix !

The Hubzero codebase is extensive and there are a lot of features that need upkeep.

- Create Portable Code. Use relative paths, environment variables, and DNS entries (no IP addresses) to reference files and assets to ensure your project does not break when your code is migrated to another machine during maintenance.

Development Environment

Setting up your Development Environment

The HUBzero Platform is comprised of many software packages woven together into an a single system. The minimum requirements for developing HUBzero web code can be a simple Linux Apache MySQL PHP (LAMP) server matching the currently supported Linux distribution. At the time of writing, Centos 7. Although the a simple LAMP server should suffice, it should not The HUBzero maintainers try to ensure that system administrators do not have to deviate too much from the stock distribution in order to install HUBzero.

Setup Development Environment via VMware Image

The Hubzero maintainers package a VMware Image available for evaluation and development purposes available for download at https://hubzero.org/download.

Setup Development Environment via Packages

If you are confident in your Linux system administration skills, the Hub can be built via Debian and RedHat packages. Instructions can be found in our documentation under System Administrators. Once the hub is operational, you will want to clone the hubzero-cms repository for development.

See the Getting the Latest Code section for instructions on how to use git to grab the latest code.

Getting the Latest Code

In the pre-packaged development environments listed above, the hubzero-cms code is a snapshot of when the package was created. To obtain the latest code, we will use git to clone the hubzero-cms repository.

In order to get the latest code you will need:

- SSH access to the machine
- sudo privileges
- git

A couple conventions for this section, in particular:

- this text indicates an incantation to be typed in the terminal.
- <something> indicates something that should be replaced with value you have set.
- [!] indicates something that should be interpreted, or have your judgment applied.

1. ssh <username>@<hostname>
    1. Log into your development server
2. cd /var/www [!]
    1. Change into the webroot directory.
    2. [!]: This may be /www in some configurations.
3. mv <hubname> <hubname>.bak [!]
    1. Backup the existing hubzero-cms installation.
    2. [!]: The name of the directory may change depending on what <hubname> is used for *hzcms install*. In the documentation, 'example' is used. Therefore the directory will be /var/www/example.
4. git clone https://github.com/hubzero/hubzero.git <hubname>
    1. This will clone the latest release into your webroot. The 'dev' branch is the default. You may switch to another available at this step if desired.
    2. This will create the shell of the application, we will need to (re)-populate it with configuration values and pull in external dependencies later on.
5. cp <hubname>.bak/app <hubname>/.
    1. This moves the configuration files back into the application. If prompted, overwrite the new app/ directory.
6. cd <hubname>/core
    1. Navigate to the core directory.
7. php ./bin/composer install
    1. Pull in external dependencies using composer.
8. cd ../
    1. Navigate back to the /var/www/<hubname> directory.
9. php muse migration -i -f
    1. Run database migrations. This helps maintain the database's schema between versions. Note: Reversing database schema changes can be extremely difficult if needed.
10. cd /var/www
    1. Navigate back to the directory containing the hub installation.
11. chown apache:apache <hubname> -R [!]
    1. Change the owner and the group of the hub installation directory.
    2. [!] The user may be *apache* OR *www-data*, depending on the distribution.
12. chmod 775 <hubname> -R
    1. Change the file permissions to allow the group to read and write permissions.
13. sudo usermod -aG apache <username>
    1. Add your user to the apache group.
14. git pull --rebase
    1. Pull the latest code. This should be done periodically.
    2. Use the rebase flag if you already have some commits that you haven't pushed yet. This puts your work on top of the changes that come downstream.

Note: [!] The permissions scheme used here is NOT production safe. It's used for development purposes. File permissions and your security scheme depend on your needs.