Extensions (general)

Overview

HUBzero CMS is already a rich featured content management system but if you're building a hub and you need extra features which aren't available by default, you can easily extend it with extensions. There are five types of extensions: Components, Modules, Plugins, Templates, and Languages. Each of these extensions handle specific functionality.

Components

The largest and most complex of the extension types, a component is in fact a separate application. You can think of a component as something that has its own functionality, its own database tables and its own presentation. So if you install a component, you add an application to your website. Examples of components are a forum, a blog, a community system, a photo gallery, etc. You could think of all of these as being a separate application. Everyone of these would make perfectly sense as a stand-alone system. A component will be shown in the main part of your website and only one component will be shown. A menu is then in fact nothing more then a switch between different components.

Modules

Modules are extensions which present certain pieces of information on your site. It's a way of presenting information that is already present. This can add a new function to an application which was already part of your website. Think about latest article modules, login module, a menu, etc. Typically you'll have a number of modules on each web page. The difference between a module and a component is not always very clear for everybody. A module doesn't make sense as a standalone application, it will just present information or add a function to an existing application. Take a newsletter for instance. A newsletter is a module. You can have a website which is used as a newsletter only. That makes perfectly sense. Although a newsletter module probably will have a subscription page integrated, you might want to add a subscription module on a sidebar on every page of your website. You can put this subscribe module anywhere on your site.

Another commonly used module would be a search box you wish to be present throughout your site. This is a small piece of re-usable HTML that can be placed anywhere you like and in different locations on a template-by-template basis. This allows one site to have the module in the top left of their template, for instance, and another site to have it in the right side-bar.

Plugins

Plugins serve a variety of purposes. As modules enhance the presentation of the final output of

the Web site, plugins enhance the data and can also provide additional, installable functionality. Plugins enable you to execute code in response to certain events, either core events or custom events that are triggered from your own code. This is a powerful way of extending the basic functionality.

Templates

A template is a series of files within the Joomla! CMS that control the presentation of the content. The template is not a website; it's also not considered a complete website design. The template is the basic foundation design for viewing your website. To produce the effect of a "complete" website, the template works hand-in-hand with the content stored in the database.

Each hub comes with default templates for both the administrator area and the front-end site.

Languages

Probably the most basic extensions are languages. Languages can be packaged in two ways, either as a core package or as an extension package. In essence, these files consist key/value pairs, these pairs provide the translation of static text strings which are assigned within the source code. These language packs will affect both the front and administrator side. Note: these language packs also include an XML meta file which describes the language and font information to use for PDF content generation.

Conclusion

If the difference between the three types of extensions is still not completely clear, then it is advisable to go to the admin pages of your installation and check the components menu, the module manager and the plugin manager. A hub comes with a number of core components, modules and plugins. By checking what they're doing, the difference between the three types of building blocks should become clear.

Installing

Installing From Package

Warning: Most HUBs do **not** have public write access to the various extensions directories. Using this method may fail as a result. Contact your system administrator for any necessary changes.

The CMS provides a convenient Installer utility in the administrative back-end. From here, one can install new modules that have been packaged as .zip files. The installer moves all the necessary files to their appropriate locations and creates any database entries needed.

Note: There is usually an upper limit to the size of files that can be uploaded within the web server itself. This limit is set in the PHP configuration file and may differ between web servers and hosts. Current HUB installs set the limit to **100MB**. This limit cannot be altered from within the CMS. Contact your system administrator for help if needed.

1. Log in to the administrative back-end of the HUB you wish to install the module on.
2. Once logged-in navigate to the Extensions Installer. This can be found from the main menu by following the "Install/Uninstall" option found in the drop-down under "Extensions".
3. Under "Upload Package File", click on the "Browse" (note: some browsers/OSes may have alternate wording) button. This will open the File Upload dialogue window. Navigate to the location of the desired package file on the local hard drive. Select the extension file and click the Open button. The dialogue window will disappear and the path to, and name of, the extension file will appear in the File Upload field.
4. Click the "Upload File & Install" button to complete the transfer and installation of a copy of the extension files from the local computer to the /yourhub/app/{ExtensionType}/ directory tree.

Installing From Directory

The CMS provides a convenient Installer utility in the administrative back-end. From here, one can install new modules from an existing directory on the server. The installer moves all the necessary files to their appropriate locations and creates any database entries needed.

1. If the module is packaged as a .zip file, unpack it onto the local hard drive before uploading.
2. Upload the entire contents of the module via SSH/sFTP. Ideally the file should be transferred to the /www/yourhub/app/extensions/yourmodulename directory.

   See Accessing Files for further details on how to use SSH/sFTP.

3. Log in to the administrative back-end of the HUB.
4. Once logged-in navigate to the Extensions Installer. This can be found from the main menu by following the "Install/Uninstall" option found in the drop-down under "Extensions".
5. Under "Install from Directory" enter the exact location of the module file (it must be the **absolute** location) in this example: /www/yourhub/app/extensions/yourmodulename.
6. Click the "Install" button to complete the installation. The appropriate module files will be moved to the /yourhub/app/modules/ directory tree.

Installing By Hand

Installing an extension by hand requires a few more steps than the Extensions Installer but is still a fairly easy and quick process.

1. If the extension is packaged as a .zip file, extract the files to a location on your local machine.

2. Upload the entire contents of the extension, except language files, via SSH/sFTP to the /yourhub/app/{ExtensionType}/ directory.

Extension Type	Install Location
Component	/yourhub/app/components/{ExtensionName}
Module	/yourhub/app/modules/{ExtensionName}
Plugin	/yourhub/app/plugins/{PluginType}/{PluginName}
Template	/yourhub/app/templates/{ExtensionName}

   See Accessing Files for further details on how to use SSH/sFTP.

3. Log in to the administrative back-end of the HUB.

4. **Components**

   1. Components do not technically need a database entry to function in their simplest form. However, an entry is needed if one wishes to use parameters or have the component appear under the "Components" list in the administrative back-end. This must be done by hand via MySQL command-line, some form of MySQL database GUI, or executing a PHP script. A sample SQL is provided below:

      ```
      INSERT INTO #__extensions(
       `extension_id`,
      ```

```
 `name`,
 `type`,
 `element`,
 `folder`,
 `client_id`,
 `enabled`,
 `access`,
 `protected`,
 `manifest_cache`,
 `params`,
 `custom_data`,
 `system_data`,
 `checked_out`,
 `checked_out_time`,
 `ordering`,
 `state`
)
VALUES(
 '',
 'com_mycomponent',
 'component',
 'com_mycomponent',
 '',
 0,
 1,
 1,
 0,
 '',
 '',
 '',
 '',
 0,
 '0000-00-00 00:00:00',
 0,
 0
);
```

See Direct Database Access for further details on how to access a HUB's database via command-line or GUI utility.

Modules

1. Once logged-in navigate to the Modules Manager. This can be found from the

main menu by following the "Modules Manager" option found in the drop-down under "Extensions".
2. Click the "New" button in the toolbar. This will present you with a list of all available modules, including those with existing directories but no database entries (such as the one you just copied to /yourhub/app/modules/).
3. Find the name of your newly added module and click its radio button. Once selected, click the "Next" button in the toolbar. This will take you to an "edit module" screen where you may enter a title, adjust parameters, select a position, etc.
4. Enter a title, adjust parameters, select a position, and enter any other necessary information. Click "Save" in the toolbar.

Plugins

1. Unlike modules, there is no convenient Joomla! utility to create the necessary database entry for us. This must be done by hand via MySQL command-line, some form of MySQL database GUI, or executing a PHP script. A sample SQL is provided below:

```
INSERT INTO #__extensions(
 `extension_id`,
 `name`,
 `type`,
 `element`,
 `folder`,
 `client_id`,
 `enabled`,
 `access`,
 `protected`,
 `manifest_cache`,
 `params`,
 `custom_data`,
 `system_data`,
 `checked_out`,
 `checked_out_time`,
 `ordering`,
 `state`
)
VALUES(
 '',
 'System - Hello World',
 'plugin',
 'helloworld',
 'system',
 0,
 1,
 1,
```

```
 0,
 '',
 '',
 '',
 '',
 0,
 '0000-00-00 00:00:00',
 0,
 0
);
```

See Direct Database Access for further details on how to access a HUB's database via command-line or GUI utility.

Templates

1. Once logged-in navigate to the Templates Manager. This can be found from the main menu by following the "Template Manager" option found in the drop-down under "Extensions".
2. Here you will be presented with a list of available templates. Your newly added template should be available. To make it the default template of the site, select it by clicking the radio button next to its name.
3. Click the "Default" button to make the template the default.

Parameters

Overview

Standard parameter types

There are 21 different standard parameter types supported for all extension types (templates, components, modules and plugins). This section gives a brief description of each parameter type, in alphabetical order. Full details of each parameter type are given on the following pages.

- **calendar** provides a text box for entry of a date. An icon next to the text box provides a link to a pop-up calendar, which can also be used to enter the date value.
- **category** provides a drop down list of categories from a given section.
- **editors** provides a drop down list of the available WYSIWYG editors.
- **filelist** provides a drop down list of files from a specified directory.
- **folderlist** provides a drop down list of folders from a specified directory.
- **helpsites** provides a drop down list of the help sites for your Joomla installation.
- **hidden** provides a hidden field for saving a parameter whose value cannot be altered directly by a user in the Administrator (it can be altered in code or by editing the *params.ini* file).
- **imagelist** provides a drop down list of image files in a specified directory.
- **languages** provides a drop down list of the installed languages for the Front-end or Back-end.
- **list** provides a drop down list of custom-defined entries.
- **menu** provides a drop down list of the available menus from your Joomla site.
- **menuitem** provides a drop down list of the available menu items from your Joomla site.
- **password** provides a text box for entry of a password. The password characters will be obscured as they are entered.
- **radio** provides radio buttons to select different options.
- **spacer** provides a visual separator between parameter field elements. It is purely a visual aid and no parameter value is stored.
- **sql** provides a drop down list of entries obtained by running a query on the Joomla Database. The first results column returned by the query provides the values for the drop down box.
- **text** provides a text box for data entry.
- **textarea** provides a text area for entry of multi-line text.
- **timezones** provides a drop down list of time zones.
- **usergroup** provides a drop down list of user groups.

Uninstalling

Overview

If you wish to uninstall an extension on your site, then follow these simple steps:

1. Select "Extensions" and then "Install / Uninstall" from the drop-down menu
2. Select the type of extension you wish to uninstall. You will have the choice between Components, Modules, Plugins, Languages and Templates.
3. Find the extension you wish to uninstall and check the checkbox to the left of the extension title.
4. In the upper-right corner of the screen, press "Uninstall"

It's as simple as that. If the CMS can't uninstall the extension, you will be prompted with an error message. If this happens, it's most likely to be caused by the extension. As extensions are developed by third-party volunteers, you will have to try to get support from the developers of the specific extension.

Languages

Overview

To create your own language file it is necessary that you use the exact contents of the default language file and translate the contents of the define statements. Language files are INI files which are readable by standard text editors and are set up as key/value pairs.

Working With INI Files

INI files have several restrictions. If a value in the ini file contains any non-alphanumeric characters it needs to be enclosed in double-quotes ("). There are also reserved words which must not be used as keys for ini files. These include: NULL, yes, no, TRUE, and FALSE. Values NULL, no and FALSE results in "", yes and TRUE results in 1. Characters {}|&~![()" must not be used anywhere in the key and have a special meaning in the value. Do not use them as it will produce unexpected behavior.

Files are named after their internationally defined standard abbreviation and may include a locale suffix, written as language_REGION. Both the language and region parts are abbreviated to alphabetic, ASCII characters. A user from the USA would expect the language English and the region USA, yielding the locale identifier "en_US". However, a user from the UK may expect a region of UK, yielding "en_UK".

Setup

As previously mentioned, language files are setup as key/value pairs. A key is used within the widget's view and the translator retrieves the associated string for the given language. The following code is an extract from a typical widget language file.

```
; Module - Example (en_US)
MOD_EXAMPLE_HERE_IS_LINE_ONE = "Here is line one"
MOD_EXAMPLE_HERE_IS_LINE_TWO = "Here is line two"
MOD_EXAMPLE_MYLINE = "My Line"
```

Translation keys can be upper or lowercase or a mix of the two and may contain underscores but no spaces. HUBzero convention is to have keys all uppercase with words separated by underscores, following a pattern of {ExtensionPrefix}_{WidgetName}_{TextName} for naming.

Table 1: Translation key prefixes for the various extensionsExtension Type Key Prefix
Component

Extension Type	Key Prefix
Module	
Plugin	
Template	

Adhering to this naming convention is not required but is strongly recommended as it can help avoid potential translation collisions. Since a component can potentially have modules loaded into it, the possibility of a widget and a module having the same translation key arises. To illustrate this, we have the following example of a component named mycomponent that loads a module named mymodule.

The language files for both:

```
; mymodule en_US.ini
MYLINE = "Your Line"
```

```
; mycomponent en_US.ini
MYLINE = "My Line"
```

The layout files for both:

```
<!-- mymodule layout -->
<strong><php echo Lang::txt('MYLINE'); ?></strong>
```

```
<!-- mycomponent layout -->
<div>
 <!-- Load the module -->
 <php echo Module::byPosition('mymodule'); ?>
 <!-- Translate some component text -->
 <php echo Lang::txt('MYLINE'); ?>
</div>
```

Outputs:

```
<div>
 <!-- Load the module -->
 <strong>Your Line</strong>
 <!-- Translate some component text -->
 Your Line
</div>
```

Since the module is loaded in the component view, i.e. *after* the component's translation files have been loaded, the module's instance of MYLINE overwrites the existing MYLINE from the component. Thus, the view outputs "Your Line" for the component translation instead of the expected "My Line". Using the HUBzero naming convention of adding component and module name prefixes helps avoid such errors:

The language files for both:

```
; mymodule en-US.ini
MOD_MYMODULE_MYLINE = "Your Line"
```

```
; mycomponent en-US.ini
COM_MYCOMPONENT_MYLINE = "My Line"
```

The view files for both:

```
<!-- mymodule view -->
<strong><php echo Lang::txt('MOD_MYMODULE_MYLINE'); ?></strong>
```

```
<!-- mycomponent view -->
<div>
 <!-- Load the module -->
 <php echo $this->Widgets()->renderWidget('mywidget'); ?>
 <!-- Translate some module text -->
 <php echo Lang::txt('COM_MYCOMPONENT_MYLINE'); ?>
```

```
</div>
```

Outputs:

```
<div>
 <!-- Load the widget -->
 <strong>Your Line</strong>
 <!-- Translate some module text -->
 My Line
</div>
```

To Further avoid potential collisions as it is possible to have a component and module with the same name, module translation keys are prefixed with MOD_ and component translation keys with COM_.

Translating Text

A translate helper (Lang) is available in all views and the appropriate language file for an extension is preloaded when the extension is instantiated. This is all done automatically and requires no extra work on the developer's part to load and parse translations.

Below is an example of accessing the translate helper:

```
<p><?php echo Lang::txt("MOD_EXAMPLE_MY_LINE"); ?></p>
```

Strings or keys not found in the current translation file will output as is.