0.8.0

HUBzero 0.8.0

Hub Users

This documentation is out-of-date, please visit http://hubzero.org/documentation/current for the latest release.

The User Guide is a user manual for a HUB. It progresses step-by-step through the features of the site and how a user can accomplish such tasks as:

- Edit their profile
- Start and manage a user group
- Participate in the Questions & Answers forum
- Submit, comment on, and monitor support tickets
- Submit a new resource
- And more...

Tutorials

How-To Videos

The HUBzero team is proud to present a growing series of How-to videos hosted on YouTube.

We've used our flagship site, nanoHUB.org, to demonstrate basic HUB concepts; however, most other HUB sites should function similarly, if not identically, to nanoHUB. Below is an up-to-date list of How-to videos, organized by topic. To view the entire series, click the YouTube icon in the upper-right corner of this page.

{xhub:module position="ytplaylistintro" style="-2"} {xhub:module position="ytplaylisttools" style="-2"} {xhub:module position="ytplaylistgroups" style="-2"} {xhub:module position="ytplaylistresources" style="-2"}

Topics & Wiki Pages

Creating a New Topic/Wiki Page

To create a new topic page:

Note: The Page text box uses wiki markup formatting. Plain text (no wiki markup) is allowed, but if you want to apply font style changes, embed images/files, add links, etc., you will have to use wiki markup. The toolbar on top displays shortcut buttons for common wiki markup commands. When you click any of the icons, the command is automatically provided and all you have to do is type the words you want to be formatted in place of the highlighted text within the command.
icon. You will see this command where you had
. Replace the words "Headline text" with what you want for the heading.

link at the bottom of the Page text box.

For instance, to create a heading, click on the

the cursor:

If you need more help with wiki formatting, click on the

1. Log in on the hub's front-end interface.
2. Navigate to **/topics**.
3. Click on **New Page**.

4. Fill in the required fields.

5. Click **Preview** if you'd like to see how your page looks like before submitting it.
   Click **Submit** when ready.

   The topic page is now created.

   Note: If the main topics page is using a macro that displays all existing topics pages on the hub, your new page will be added to the list on the main topics page. If not, your new page will have to be manually added to the list so that it is accessible to everybody. If you know the URL of your topics page (which, by default, is the title you have set for your page, only with no spaces), you can get to it by typing it on the address bar in this format: https://yourhub.org/topics/NAME_OF_YOUR_PAGE. For example, a topics page with the title "My New Page" will have the URL: https://yourhub.org/topics/MyNewPage.

To create a new wiki page within a group:

Note: If the main group wiki page is using a macro that displays all existing wiki pages within the group, your new page will be added to the list on the main group wiki page. If not, your new page will have to be manually added to the list so that it is accessible to everybody. If you know the URL of your group wiki page (which, by default, is the title you have set for your page, only with no spaces), you can get to it by typing it on the address bar in this format: https://yourhub.org/groups/NAME_OF_YOUR_GROUP/wiki/NAME_OF_YOUR_PAGE. For example, a group wiki page with the title "My New Page" will have the URL: https://yourhub.org/groups/samplegroup/wiki/MyNewPage.

1. Go to the **Wiki** tab inside a group.

2. Click on **New Page**.

3. Fill in the required fields.

4. Click **Preview** if you'd like to see how your page looks like before submitting it. Click **Submit** when ready.

Editing Content on Topics/Wiki Pages

To edit wiki content:

1. Open the topic/wiki page you would like to edit.

2. Go to the Edit section of the topic/wiki page.

3. Edit the page.

4. When you're done with all your changes, scroll down the bottom of the page and click **Preview** if you'd like to see how your page looks like before submitting it. Click **Submit** when ready.

The topic/wiki page is now updated.

Uploading an Image or File to Topics/Wiki Pages

In order to embed an image or file on the topic/wiki page, you must first upload the image or file in the wiki.

To upload an image or file:

1. Go to the Edit section of the wiki page you would like to edit.

2. To the right of the page, you will see a box for uploading a file. Click on the **Browse** button.

3. Locate the image or file you would like to upload and click **Open**.

4. Click **Upload**.

Your file is now uploaded in the wiki and you can now embed it in the wiki's Page text box so others can view or download it.

Embedding an Image or File on Topics/Wiki Pages

Note: In order to embed an image or file on the topic/wiki page, you must first upload the image or file.

To embed an uploaded image/file:

1. Go to the Edit section of the wiki page you would like to edit.

2. On the Page text box, place your cursor where you want the image or file to be. If you are uploading an image, click the **Embedded Image** button on the toolbar. If you are uploading a file, click on the **Embedded File** button.

3. Replace the highlighted text with the full name of the uploaded item you want to embed.

   For image:

4. Scroll down to the bottom of the page and click **Submit**.

The item you chose to embed will now appear in the area where you placed it on the wiki. If you embedded a file, it will show up as a link that users can download.

Making a List of Available Topic Pages/Wikis

When a new topic/wiki page is created, there is really no way to access the newly created wiki unless you know the URL of it (see Creating a Topic/Wiki Page tutorial to figure out the URL of your topic/wiki page). *A wiki macro (or custom function to insert dynamic content in a page) can be added to a page to display a list of existing topic pages in Topics or wiki pages within a group.*

To add a wiki macro that generates a list of all topic/wiki pages:

1. Go to the Edit section of the topic/wiki page you would like to edit.

2. Type in **[[TitleIndex]]** where you want the list to appear

3. Click **Submit**.

You will then see a bulleted list of all available pages where you added the macro.

Hub Managers

The Managers' Guide is a user manual for managing the web content and functionality of a HUB. It progresses step-by-step through various common tasks and familiarizes the manager with the administrative back-end interface. Managers will learn how to accomplish tasks such as:

- Install new modules/widgets/plugins/templates
- Manage (add/edit/delete) site members
- Manage site settings
- Handle support tickets
- And more ...

Introduction

Overview

The Managers' Guide is a user manual for managing the web content and functionality of a HUB. It progresses step-by-step through various common tasks and familiarizes the manager with the administrative back-end interface. Managers will learn how to accomplish tasks such as:

- Install new modules/widgets/plugins/templates
- Manage (add/edit/delete) site members
- Manage site settings
- Handle support tickets
- And more ...

We strongly recommend reviewing Joomla!'s Documentation for further help on topics that may not be covered here.

How To Use The Guide

The guide is grouped by Subject, then Chapters within a subject, then Sections within Chapters.

Administrator Area

Logging In

The Administrator back-end of your Joomla! installation is where you do most of the work to set up, configure and maintain your website. As such it must be protected from access by casual visitors and so Joomla! will request a valid username and password before access to the Administrator is granted.

You access the Administrator back-end of your website by entering a special URL into the address bar of your web browser. If your website URL is http://www.example.com then you will enter the address http://www.example.com/administrator. You will be shown the Administrator login form which looks like this:

Enter your username and password into the relevant fields. If required, select your preferred language from the drop-down list. Then click the "Login" button. If you are logged in successfully you should see the Administrator control panel screen which looks similar to this:

Logging Out

To log out of the Administrator back-end, locate the "Logout" link in the top-right corner of the Administrator screens and click on it. Logging out will return you to the Administrator login screen. If the "Logout" link is grayed out and nothing happens when you click on it, then you are probably in a screen where you have some resource locked for your exclusive use. Click the "Cancel" toolbar button first to release the resource, then click on the "Logout" link.

FAQ

Customizing Your New Hub

The Front Page

- Edit the "welcome" area on the home page
- Change the rotating banner on the home page
- Change text or other components on the home page

Registration

- Change the information new users must supply when registering
- Change the text of the confirmation email
- Change the URL to redirect to after confirming email

Login

- Change the URL to redirect to after logging in

Content

- Edit "about" pages, terms of use, etc.
- Add a picture to an article/page
- Change the types of resources that users can contribute
- Change categories of events on the public calendar
- Change metadata exposed to search engines

How do I ...

Content

- ...change items in the menu bar along the top of the site?
- ...add a new page to the site?
- ...fix spelling errors on a resource that a user has uploaded?
- ...help a user upload a huge resource file?
- ...fix spelling errors on tags, or delete useless tags?
- ...put up a poll question for the community?

Maintenance

- ...put up a notice that the site is going down for maintenance?

Members

- ...find the login or email address for a registered user?
- ...make another user a site administrator?
- ...award points to a user that did something special to help?

Daily Maintenance

- Keep an eye on the dashboard
- Approve pending resources
- Support tool contributions
- Handle support tickets
- Respond to reports of abuse
- Fulfill orders in the site store

Media Manager

Overview

See Joomla!'s documentation.

Configuring

Overview

One of the many advantages to Joomla! and HUBzero is the level of configuration that can be applied. Not only can you configure some global site settings, but individual components, modules, and plugins as well.

HUBzero also adds many other configuration options for extended registration information and more.

Joomla! Settings

Overview

Note: Most, if not all, of these settings can be set once and then left alone. Most HUBzero installs will have these configurations set for optimal performance. It is not advisable to make changes to the "System" or "Server" settings.

The settings are saved in '/configuration.php'. To save your changes, either the FTP-layer must be activated or the 'configuration.php' file made writable.

Toolbar

At the top right you will see the toolbar. The functions are:

Save
: Save the modifications you made to the Global Configuration. You will be redirected to the Control Panel

Apply
: Save it, but stay in the same screen. If you have been working on a screen for a long time and don't want to risk losing your work, pressing Apply saves your work and lets you continue working. If, for example, you lost your Internet connection, your work will be saved up this point.

Close
: Return to the previous screen without saving your work. If you press Close while adding a new item, this new item will not be created. If you were modifying an existing item, the modifications will not be saved.

Help
: Opens the Joomla! Help Screen.

Site Setting Groups

Site Settings

Site Offline
: This setting shows when the site is offline. Only Administrators will be able to see the site when *Site Offline* is set to Yes. The default setting is **No**.

Offline Message
: The message that will be displayed on the site when the site is offline.

Site Name
: The name of the site.

Default WYSIWYG Editor
: (WYSIWYG stands for What You See Is What You Get) The default editor to use when creating articles.

List Length
: The length of lists in the Control Panel for all Users. By default, this is set to **20**.

Feed Length
: The number of content items to be shown in the feed(s). By default, this is set to **10**.

Metadata Settings

Global Site Meta Description
: This is the description of the site which is indexed by search engine spiders.

Global Site Meta Keywords
: These keywords describe the site and are the basis for improving the ability of search engine spiders ability to index the site.

Show Title Meta Tag
: It shows the Meta information of each article. This Meta information is used by search engine spiders when indexing the site. Each article can have its own Meta Data information (set under the **Metadata Information** pane when creating or editing an article).

Show Author Meta Tag
: It shows the Author Meta information for articles and is used by search engine spiders when indexing the site.

SEO Settings

SEO stands for *Search Engine Optimization*.

Search Engine Friendly URLs
: When set to *Yes*, URLs are rewritten to be more friendly for search engine spiders. For example, the URL: *www.example.com/index.php?option=com_content&view=etc...*, would turn into: *www.example.com/alias*. Most of the items created in Joomla! have an Alias box where a search engine friendly URL can be inserted. The default setting is **No**.

Use Apache *mod_rewrite*
: When set to *Yes*, Joomla! will use the *mod_rewrite* settings of Apache when creating search engine friendly URLs. Please note: it is advised that you do not modify any **.htaccess** file without an understanding of how it works. You must use the **.htaccess**

file provided with Joomla! in order to use this setting. To use this file, rename the **htaccess.txt** file (found in the root directory) to **.htaccess**. By default, this setting is set to **No**.

Add suffix to URLs
: When set to *Yes*, Joomla! will add **.html** to the end of the URLs. The default setting is **No**.

System Setting Groups

System Settings

Secret Word
: This is generated when Joomla! is first installed and is not changeable. It is used internally by Joomla! for security purposes.

Path to Log folder
: The path where the logs should be stored. The Joomla! installer should automatically fill in this folder.

Enable Web Services
: This feature enables Joomla! to make RPC (Remote Procedure Calls) using HTTP as the transport medium and XML as the encoding language. This function should remain on to ensure that many third party extensions works. The default setting **Yes**.

Help Server
: The place Joomla! looks for help information when you click the **Help** button (visible in many screens and options of the administration panel). By default, it uses Joomla!'s main help site.

User Settings

Allow User Registration
: This determines whether users can register on the site. The default setting is **Yes**.

New User Registration Type
: If *Allow User Registration* is set to *Yes*, this setting tells Joomla! what level of access should be given to new users who register. The default setting is **Registered**.

New User Account Activation
: This setting determines whether a new user will have to activate their account before they can use it. If set to **Yes**, users will be sent an email containing a link to a web address. The user must follow this link to activate his/her account. By default, this setting is set to **Yes**.

Front-end User Parameters
: When set to *Show*, users will be able to select their language, article editor, and help site preferences from within their Details screen. The default setting is **Show**.

Media Settings

Legal Extensions (File Types)
: These are the file types users are allowed to upload. By default, when Joomla! is installed, basic image and document files are allowed.

Maximum Size (in bytes)
: The maximum file size users are allowed to upload, in bytes. The default setting is **10000000** (or 10mb).

Path to Media Folder
: The path where non-image media files are located, including videos and document files. The default is "<Joomla! home>/images". You can create a new folder to use with the Media Manager and enter the path to that folder here. **Warning**: *Do not delete or rename the existing "<Joomla! home>/images" folder on your server.* This folder and its subfolders "images/banners", "images/M_images", "images/smiles", and "images/stories" are used by Joomla!. Deleting or renaming these folders may cause errors.

Path to Image Folder
: The path where images are stored. The default is "<Joomla! home>/images/stories". You can create a new folder to use with the Media Manager and enter the path to that folder here. If you want to access this folder from the Media Manager, either use the default value or make the new folder a sub-folder of the Media Manager folder. **Warning**: *Do not delete or rename the existing "images/stories" folder on your server.* This folder is used by Joomla! and deleting or renaming it will cause errors.

Restrict Uploads
: This feature restricts uploads by user type. The default setting is **Yes**.

Check MIME Types
: This uses MIME Magic or Fileinfo to verify files. The default setting is **Yes**. By checking the MIME information, you help ensure users don't upload malicious files to your site. If invalid MIME type errors are received then change the setting to *No*.

Legal Image Extensions (File Types)

This allows you to limit the types of images that can be uploaded. It operates by checking the file image headers. By default, Joomla! only allows **bmp,gif,jpg,png**.

Ignored Extensions
: This sets which extensions are ignored for MIME type checking and restricts uploads. By default, no extensions are ignored.

Legal MIME Types
: This sets the list of legal MIME types for uploads. By default, Joomla! automatically includes some standard file types. It is recommended that you do not touch this setting unless you know what you are doing.

Illegal MIME Types
: This sets the list of illegal MIME types for uploads. By default, Joomla! automatically blocks HTML MIME types. It is recommended that you do not touch this setting unless you know what you are doing.

Enable Flash Uploader
: This setting enables the integrated Flash Uploader which is used for example at the Media Management. If enabled it is possible to upload several fieles at once. Default setting is **No**. ***Tip****: if enabled and the download does not work, disable the flash uploader (can happen due incompatible flash settings by Adobe)*

Debug Settings

Debug System
: This will turn on the debugging system of Joomla!. When set to **Yes**, this tool will provide diagnostic information, language translations, and SQL errors. If any such issues or errors occur, they will be displayed at the bottom of each page, in both the front-end and back-end.

Debug Language
: This will turn on the debugging indicators (*...*) or (?...?) for the Joomla! Language files. Debug Language will work without the Debug System tool set to on. But it will not provide additional detailed references which would help in correcting any errors.

Cache Settings

Cache
: This setting sets whether site caching is enabled or not. When enabled, it keeps a local copy of the content on the server to speed up accessing and lessen stress on the database. The default setting is **No**.

Cache Time
: This setting sets the maximum length of time (in minutes) for a cache file to be stored before it is refreshed. The default setting is **15** minutes.

Cache Handler
: This setting sets how the cache operates. There is only one caching mechanism which is file-based.

Session Settings

Session Lifetime
: This setting sets how long a session should last and how long a user can remain signed in for (before logging them off for being inactive). The default setting is **15** minutes.

Session Handler
: This setting sets how the session should be handled once a user connects and logs into the site. The default setting is set to **Database**.

Server Setting Groups

Server Settings

Path to Temp-folder
: The path where files are temporarily stored. This is filled in by default when Joomla! is installed.

GZIP Page Compression
: Compressing pages typically increases your site's speed. The default setting is **No**.

Error Reporting
: This sets the appropriate level of reporting. The default setting is **System Default**.

Force SSL
: This setting forces the site access for selected areas under SSL (https). *Note*: you must have set already the server to use SSL. Options are:

  None
  : SSL is not activated

  Administrator Only
  : SSL is only valid for the backend.

Entire Site
: SSL is valid for the whole site (front- & backend).

Locale Settings

Time Zone
: This tool sets the current date and time. The set time should be where the site's server is located. The default setting is **(UTC 00:00) Western Europe Time, London, Lisbon, Casablanca**.

FTP Settings

FTP stands for File Transfer Protocol. Most of these settings are set during the initial Joomla! installation.

Enable FTP
: This setting tells Joomla! to use it's built-in FTP function instead of the normal upload process used by PHP.

FTP Host
: The host server's URL connecting the FTP.

FTP Port
: The port where the FTP is accessed. The default setting is **21**.

FTP Username
: The username that Joomla! will use when accessing the FTP server. Security recommendation: create another FTP user account to access a folder where files will be uploaded to.

FTP Password
: The password that Joomla! will use when accessing the FTP server. Security recommendation: create another FTP user account to access the folder where files will be uploaded to.

FTP Root
: The root directory where files should be uploaded to.

Database Settings

These settings are set during the initial setup of Joomla! It is advised to leave these settings the way they are, unless you have a good understanding of how databases work.

Database Type
: The type of databased to be used. The default setting is **mysql**, but this can be changed during the initial setup of Joomla!.

Hostname
: The hostname where the database is located. It is typically set to **localhost** for most servers. It is possible for the hostname to be located on a different server all together.

Username
: The username to access the database.

Database
: The name of the database.

Database Prefix
: The prefix used before the actual table's name. This allows you to have multiple Joomla! installations in the same database. The default setting is **jos**_, but this can be changed during initial setup of Joomla!.

Mail Settings

The mail settings are set during the initial setup of Joomla!. These settings can be changed whenever needed.

Mailer
: This setting sets which mailer to use to deliver emails from the site. The default setting is **PHP Mail Function**. This can be changed during the initial setup of Joomla!.
  PHP Mail Function
  : This uses the mail function that is built into PHP.
  Sendmail
  : This uses the Sendmail program, which is typically used when creating HTML email forms.
  SMTP Server
  : This uses the site's SMTP server.

Mail from
: The email address used by Joomla! to send site email.

From Name
: The name Joomla! will use when sending site emails. By default, Joomla! uses the site name during the initial setup.

Sendmail Path
: The path where the Sendmail program is located. This is typically filled in by Joomla! during the initial setup. This path is only used if **Mailer** is set to **Sendmail**.

SMTP Authentication
: If the SMTP server requires authentication to send mail, set this to **Yes**. Otherwise leave it at **No**. This is only used if **Mailer** is set to **Sendmail**.

SMTP Username
: The username to use for access to the SMTP host. This is only used if **Mailer** is set to **Sendmail**.

SMTP Password
: The password to use for access to the SMTP host. This is only used if **Mailer** is set to **Sendmail**.

SMTP Host
: The SMTP address to use when sending mail. This is only used if **Mailer** is set to **Sendmail**.

HUB Settings

Overview

Note: Most, if not all, of these settings can be set once and then left alone. Most HUBzero installs will have these configurations set for optimal performance. It is not advisable to make changes to any settings not in the "Registration" area.

Toolbar

At the top right you will see the toolbar. The functions are:

Save
: Save the modifications you made to the Global Configuration. You will be redirected to the Control Panel

Close
: Return to the previous screen without saving your work. If you press Close while adding a new item, this new item will not be created. If you were modifying an existing item, the modifications will not be saved.

Site Settings Group

Site Settings

Short Name
: **Deprecated** A short name for the site to be used in various component texts.

Short URL
: **Deprecated** The URL of the site minus the "http://".

Long URL
: **Deprecated** The full URL of the site.

Support Email
: Primary email address for administrative data to be sent to.

Monitor Email
: Email address for account registration data to be sent to.

Home Dir
: The home directory for user accounts.

Forge Settings

Name
: The name for the Tool Forge area of the site.

URL
: The URL for the Tool Forge.

RepoURL
: The URL for the Tool Forge used by Repo.

LDAP Settings

MasterHost
.
SlaveHosts
.
BaseDN
.
NegotiateTLS
.
SearchUserDN
.
SearchUserPW
.
AcctMgrDN
.
AcctMgrPW
.

Registration Setting Groups

See Configuring Registration for more details.

Login Return URL

By default, members who have just logged in will be directed to the "/myhub" page. It is possible, in HUBzero, to change the redirection URL to something else.

1. First login to the administrative back-end.
2. Once logged in, find “Components” in the main menu bar located toward the top of the page. You should be presented with a drop-down menu containing a list of your installed components.

3. Choose “Hub” from the available options.

4. You should now be presented with a control panel for various settings and configurations of your site.

   Note: This is a different set of configurations than what is found under “Site” > “Global Configuration” and control HUB specific items.

   Once the page has loaded, select the “Parameters” button in the toolbar, found in the upper right-hand portion of the screen. Click it.

5. You should now be presented with a pop-up panel for various settings and configurations of your site.

   Look for the option titled "Login Return URL". This will most likely be the first item in the list.

6. Enter in the textbox the URL you wish members to be redirected to and then click "Save".

Registration

Overview

HUBs allow for considerable customization of the registration form for new members. While one HUB may present fields for filling in phone, website, and organization, another may take a more minimal approach and require only a username, password, and email address. This is all configurable.

Making Changes

1. First login to the administrative back-end.
2. Once logged in, find “Components” in the main menu bar located toward the top of the page. You should be presented with a drop-down menu containing a list of your installed components.

3. Choose “Hub” from the available options.

4. You should now be presented with a control panel for various settings and configurations of your site.

   Note: This is a different set of configurations than what is found under “Site” > “Global Configuration” and control HUB specific items.

   Once the page has loaded, select “Registration” from the sub-menu found under the “HUB Configuration” title.

5. You should now be presented with a table of available user fields and their status for a particular action. This controls what fields the user will see, must fill in (required) or can fill in (optional) depending upon which action or state they are currently in. That is, you can make the username field required for the registration page (“create” column) but may not wish for your users to be able to edit this after creation (“read only” for the “update” and “edit” columns).

   Create column
   : What the user sees on the registration page

   Proxy column
   : What columns an administrator sees or must fill in when creating an account by proxy (i.e., for someone else)

   Update column
   : What fields the user will see and/or must fill in if something has changed with what information is required at registration. An example of this would be if the “citizenship” field was, at one point, optional for registration but is now required. Setting this field to “Required” for the “Update” column will now require logged-in users to fill this information out.

   Edit column
   : What fields the user will see and can edit for their user profile

   Field Option Definitions:
   : Required = Must fill in

Optional = Can fill in
Hide = Not visible
Read only = Can view but cannot change

6. Once you feel ready to save your changes, scroll back to the top of the page and click “Save” (the icon that looks like a floppy disk) in the upper right portion of the page.

Changes take affect immediately.

Customizing Confirmation Email

All component layouts can be customized through overrides. Except for files that are provided in the "Joomla!" distribution itself, this method for customization eliminate the need for designers and developers to "hack" core files that could change when the site is updated to a new version. Because they are contained within the template, they can be deployed to the Web site without having to worry about changes being accidentally overwritten when your System Administrator upgrades the site. Some emails sent by the system, such as registration confirmations, are component layouts for just this reason of customization.

Quick Overview of Output Overrides

Layout overrides only work within the active template and are located under the /html/ directory in the template. For example, the overrides for "corenil" are located under /templates/corenil/html/.

It is important to understand that if you create overrides in one template, they will not be available in other templates. For example, "rhuk_milkyway" has no component layout overrides at all. When you use this template you are seeing the raw output from all components. When you use the "Beez" template, almost every piece of component output is being controlled by the layout overrides in the template. "corenil" is in between having overrides for some components and only some views of those components.

The layout overrides must be placed in particular way. Using "Beez" as an example you will see the following structure:

```
/templates
  /beez
    /html
      /com_register  (this directory matches the component directory n
ame)
        /emails        (this directory matches the view directory name
)
          confirm.php (this file matches the layout file name)
```

The structure for component overrides is quite simple:
/html/com_{ComponentName}/{ViewName}/{LayoutName}.php.

Customizing The Confirmation Email

To override the component layout for the registration confirmation email, copy:

```
/components/com_register/views/emails/tmpl/confirm.php  (the email sen
t if the user resends a confirmation)
/components/com_register/views/emails/tmpl/create.php  (the email sent
```

```
 upon registration)
```

to:

```
/templates/{YourTemplate}/html/com_register/emails/confirm.php
/templates/{YourTemplate}/html/com_register/emails/create.php
```

If the destination directories do not exist, create them. And then edit the contents of the file(s) you just copied.

See Template Overrides for more information on customizing layouts and CSS.

Confirmation Return URL

By default, members who have just completed the confirmation process will be directed to a "Thank you" page. It is possible, in HUBzero, to change the redirection URL to something else.

Note: This only applies to the initial page the user sees upon confirmation of their email. To customize the return URL members are redirected to upon login see Hub Configuration.

1. First login to the administrative back-end.
2. Once logged in, find “Components” in the main menu bar located toward the top of the page. You should be presented with a drop-down menu containing a list of your installed components.

3. Choose “Hub” from the available options.

4. You should now be presented with a control panel for various settings and configurations of your site.

   Note: This is a different set of configurations than what is found under “Site” > “Global Configuration” and control HUB specific items.

   Once the page has loaded, select the “Parameters” button in the toolbar, found in the upper right-hand portion of the screen. Click it.

5. You should now be presented with a pop-up panel for various settings and configurations of your site.

   Look for the option titled "Confirmation Return URL". This will most likely be the second item in the list.

6. Enter in the textbox the URL you wish members to be redirected to and then click "Save".

Components

Overview

Many components will have a set of parameters that can be configured. These parameters can determine what functionality is presented to the user, be a default set of user options (what profile fields are visible to site visitors by default, for instance), or settings needed for the component to function properly. Accessing and adjusting these parameters is quick and easy.

Note: Not all components will have configurable parameters. For those that do, the configurations apply **only** to that component and should not affect any other components, site settings, or module settings.

1. First login to the administrative back-end.
2. Once logged in, find “Components” in the main menu bar located toward the top of the page. You should be presented with a drop-down menu containing a list of your installed components.

3. Choose the component you wish configure from the available options. Here we've chosen the "Hub" component.

4. Once the page has loaded, select the “Parameters” button in the toolbar, found in the upper right-hand portion of the screen. Click it.

5. You should now be presented with a pop-up panel for various settings and configurations of your site.

6. Adjust the available parameters as needed and then click "Save". Changes take affect immediately.

0.8.0

Modules

Overview

Most modules will have a set of parameters that can be configured. These parameters can determine what functionality is presented to the user, be a default set of user options, or settings needed for the module to function properly (setting the URL to a feed for a feed reader module, for instance). Accessing and adjusting these parameters is quick and easy.

Note: Not all modules will have configurable parameters. For those that do, the configurations apply **only** to that module instance. You may have multiple instances of a module with different parameter configurations.

1. First login to the administrative back-end.

2. Once logged in, go to the "Module Manager." The Module Manager can be found by selecting "Extensions" > "Module Manager" from the drop-down menu on the back-end of your HUB installation.

3. Choose the module you wish to configure from the available list.

4. Once the page has loaded, find the “Parameters” grouping, found on the right-hand portion of the screen.

5. Adjust the available parameters as needed and then click "Save" (the icon that looks like a floppy disk) in the upper right portion of the page. Changes take affect immediately.

0.8.0

Plugins

Overview

Some plugins will have a set of parameters that can be configured. These parameters can determine what functionality is presented to the user, be a default set of user options, or settings needed for the plugin to function properly. Accessing and adjusting these parameters is quick and easy.

Note: Not all plugins will have configurable parameters. For those that do, the configurations apply **only** to that plugin and should not affect component configuration, module configuration, or site configuration.

1. First login to the administrative back-end.

2. Once logged in, go to the "Plugin Manager." The Plugin Manager can be found by selecting "Extensions" > "Plugin Manager" from the drop-down menu on the back-end of your HUB installation.

3. Choose the plugin you wish configure from the available list.

4. Once the page has loaded, find the “Parameters” grouping, found on the right-hand portion of the screen.

5. Adjust the available parameters as needed and then click "Save" (the icon that looks like a floppy disk) in the upper right portion of the page. Changes take affect immediately.

0.8.0

Extensions

Overview

Joomla! already is a rich featured content management system but if you're building a website with Joomla! and you need extra features which aren't available in Joomla! by default, you can easily extend it with extensions. There are five types of extensions for Joomla!: 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

Joomla! 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. Joomla! plugins enable you to execute code in response to certain events, either Joomla! core events or custom events that are triggered from your own code. This is a powerful way of extending the basic Joomla! 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.

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 Joomla! 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 Joomla! installation and check the components menu, the module manager and the plugin manager. Joomla! comes with a number of core components, modules and plugins. By checking what theyâ€™re doing, the difference between the three types of building blocks should become clear. You can also check out the official Joomla! extensions page. Browse through the extension categories and youâ€™ll be amazed about the extension possibilities you have for your site.

Installing

Overview

See Installing Extensions in the Web Developer's Guide for details.

Modules Manager

Overview

The Module Manager is where you add and edit Joomla! Modules. In Joomla!, Modules are used to display content and/or media around the main content.

The Module Manager can be found by selecting "Extensions" > "Module Manager" from the drop-down menu on the back-end of your HUB installation.

Module Facts:

1. All Joomla! websites require at least 1 Menu Module
2. All Other Module Types are Optional. (Examples: News, Banner, Latest News, Polls)
3. Every Menu is accompanied by a menu module. (Example mod_mainmenu)
4. Multiple occurrences of similar module types.
5. Some Modules are linked to components. For example, each Menu Module is related to one Menu component. To define a Menu in Joomla!, you need to create the Menu and Menu Items using the Menus screens and then create the Module for the Menu using this screen. Other Modules, such as Custom HTML and Breadcrumbs, do not depend on any other content.

A HUBzero installation is accompanied with over 20 Joomla! default module types and additional, HUBzero-specific modules.

Column Headers

Click on the column heading to sort the list by that column's value.

#
: An indexing number automatically assigned by Joomla! for ease of reference.

Checkbox
: Check this box to select one or more items. To select all items, check the box in the column heading. After one or more boxes are checked, click a toolbar button to take an action on the selected item or items. Many toolbar actions, such as Publish and Unpublish, can work with multiple items. Others, such as Edit, only work on one item at a time. If multiple items are checked and you press Edit, the first item will be opened for editing.

Module Name
: The name of the Module. You can click on the name to open the Module for editing.

Enabled
: A green tick or a red X showing whether the use of the component is enabled/disabled. Click the icon to toggle the item between enabled and disabled.

Order
: The order to display modules within a Position. If the list is sorted by this column, you can change the display order of modules within a Position by selecting a Position in the "Select Position" filter and then clicking the arrows or by entering the sequential order and clicking 'Save Order'.

Access Level

: Who has access to this item. Current options are:

  - Public: Everyone has access
  - Registered: Only registered users have access
  - Special: Only users with author status or higher have access

  You can change an item's Access Level by clicking on the icon in the column.

Position
: The position on the page where this module is displayed. Positions are locations on the page where modules can be placed (for example, "left" or "right"). Positions are defined in the Template in use for the page. Positions can also be used to insert a Module inside an Article using the syntax "{loadposition xxx}", where "xxx" is a unique position for the module.

Pages
: The Menu Items where this Module will be displayed. Options are "All" for all Menu

Items, "None" for no Menu Items, and "Varies" for selected Menu Items. A Module will only display on Menu Items where it is selected.

Type
: The system name of the Module. Joomla! installs 20 standard Modules. Many Joomla! Extensions contribute additional Modules.

ID
: The ID number. This is a unique identification number for this item assigned automatically by Joomla!. It is used to identify the item internally, for example in internal links. You can not change this number.

Display #
: The number of items to display on one page. If there are more items than this number, you can use the page navigation buttons (Start, Prev, Next, End, and page numbers) to navigate between pages. Note that if you have a large number of items, it may be helpful to use the Filter options, located above the column headings, to limit which items display (*where applicable*).

Toolbars

Primary Toolbar

At the top right you will see the toolbar:

The functions are:

Enable
: To enable one or more items, select them using the Checkbox and press this button. You may also toggle between Enabled and Disabled by clicking on the icon in the "Enabled" column.

Disable
: To disable one or more items, select them using the Checkbox and press this button. You may also toggle between Enabled and Disabled by clicking on the icon in the "Enabled" column.

Copy
: To copy one or more Modules, select them using the Checkbox and press this button. A new Module will be created for each selected Module. The new Module will have the name "Copy of" plus the original Module name. Note that the new copies are initially disabled.

Delete
: To delete one or more items, select them and click this button. The selected items will be deleted. **Note**: *only empty items can be deleted.*

Edit
: Select one item and click on this button to open it in edit mode. If you have more than one item selected (where applicable), the first item will be opened. You can also open an item for editing by clicking on its Title or Name.

New
: To create a new instance of a Module, press this button. You will be taken to a screen that lists all of the available Modules. See Module Manager - New/Edit for information about adding Modules.

Help
: Opens this Help Screen.

Site and Administrator Links

At the top left, above the Filter, you will see the following two links:

Site

OOpens the Site tab. This is the default tab and allows you to manage the Modules for the front end of the web site.

Administrator

OThis tab allows you to manage the Modules for the back end administration of the web site. If you do not need to change the Joomla! administrator menus, no modifications are required here.

List Filters

Filter by Partial Title

You can filter the list of items by typing part of the title or the ID number. Or, you can select a combination of Category and Published State.

Filter

In the upper left corner just above the column headings is a Filter field and two buttons, as shown below:

If you have a large number of items on the list, you can use this filter to find the desired item(s) quickly. Enter either part of the title or an ID number and press 'Go' to display the matching items. You can enter in whole words or part of a word. For example, "ooml" will match all titles with the word "Joomla!" in them.

Filter by Template, Position, Type and Published State

In the upper right area, above the column headings, are following drop-down list boxes as shown below:

The selections may be combined. Only items matching both selections will display in the list.

Select Template.
: Select the Template from the drop-down list box of available Templates. Only Templates that are enabled for this site will display.

Select Position.
: Select a Position from the drop-down list box of available Positions.

Select Type.
: Select the Module Type from the drop-down list box of available Module Types. Joomla! installs with 20 available Module Types. Additional ones may be available if you have installed any Joomla! Extensions.

Select State.
: Use the drop-down list box to select the published state: Enabled or Disabled.

Create/Edit Modules

When you create a new Module, you will select the Module Type. This will determine what type of content this Module can display. If you edit an existing Module, the Module Type will already be determined.

Note: You cannot change the Module Type of an existing Module.

All Modules have two sections that are the same: Details and Menu Assignment. The Parameters are different for each Module Type.

Details

Module Type
: This displays the system name of the module. No entry is allowed.

Title
: The Title of the Module.

Show Title
: Whether or not to show the Title of the Module.

Enabled
: Whether or not the Module is enabled. If "No", the Module will not be shown on the page.

Position
: The Position on the page to show the Module. There are two types of Positions you can use.

    1. You can select a Position from the drop-down list box. The Positions in the list box are those provided by your Template. You can see the available Positions on the page by adding the letters "?tp=1" to the end of any front-end Joomla! URL. For example, if your home page is "www.mysite.com", you can type the URL "www.mysite.com?tp=1". The page will display with labeled rectangles

indicating each pre-defined Position for the current Template.
2. You can type in a Position that is different from the pre-defined positions (for example, "myposition1"). This type of Position can be used to insert a Module into an Article. If you enter the code "{loadposition xxx}" inside the text of an Article, where "xxx" matches the Position of a Module, and if the Module is assigned to the the Menu Selection where the Article is displayed (see Menu Assignment below), then the Module will be displayed at that point in the Article.

Order

The Order that this Module will be displayed within the Position. If more than one Module is displayed at the same position, this will determine the order of display. The drop-down list box shows all Modules displaying at the current Position. Select the Module that you want the current Module to *follow*.

- You can also change the order of display in the Module Manager.
- The Module Order is only important when two or more Modules are displaying at the *same Position* and on the s*ame Menu Selections*.

Access Level. Who has access to this item. Current options are:

- Public: Everyone has access
- Registered: Only registered users have access
- Special: Only users with author status or higher have access

Enter the desired level using the drop-down list box.

ID

The ID number. This is a unique identification number for this item assigned automatically by Joomla!. It is used to identify the item internally, for example in internal links. You can not change this number.

Description

A description of what the Module does. No entry is allowed.

Menu Assignment

This is where you tell Joomla! which pages to show this Module on. This Module will show on the page when this Menu Item is selected. If the Menu Item allows the user to navigate to sub-pages, then this Module will also show on these sub-pages. For example, if the Menu item is a Blog Layout, the Module will also show when the user clicks on a "Read more..." link, an Article Link, or a Page Navigation link from that Blog Layout.

Menus.
: "All" means show the current Module on all Menu Items. "None" means don't show the Module on any Menu Items. "Select Menu Item(s) from the List" allows you to select which Menu Items to show the Module on.

Menu Selection.
: If "Select Menu Items(s) from the List" above is selected, select individual Menu Items that will display the Module.
  - Use Ctrl+Click to select multiple Menu Items.
  - Use Shift+Click to select a range of Menu Items.
  - If you want to select most of the Menu Items, first click "All" above under Menus and then click "Select Menu Items(s) from the List". All of the Menu Items will be selected. Then use Ctrl+Click to deselect the Menu Items you don't want to include.

Parameters

This will vary between modules.

Templates Manager

Overview

The Template Manager is where you assign a default Template to your Joomla! web site. You can also edit and preview Templates here.

In Joomla!, the visual layout of both the Front-end and Back-end of your site is controlled by the Template. Templates are extensions that contain layout and style information that tells Joomla! exactly how to draw each page of your site.

When you first install Joomla!, one Back-end Template and two Front-end templates are included. Other Templates can be installed from third-party developers as Extensions.

If you want to use the same Template for all of the pages on your site, you just assign one Template as the Default Template. You can also assign different Templates to different pages.

Select "Extensions" ? "Template Manager" from the drop-down menu in the back-end of your Joomla! installation.

Column Headers

#
: An indexing number automatically assigned by Joomla! for ease of reference.

Template Name
: The name given to each Template by the Template author. Click the Name to open the Template for editing. If you hover the mouse over the Template Name, a small preview for the Template displays in a pop-up window. The Template Name normally corresponds to the sub-directory name that contains the Template in the <path-to-Joomla!>/templates/ directory. For example, the files for the "rhuk_milkyway" Template are in the directory "<path-to-Joomla!>/templates/rhuk_milkyway".

Default
: Indicator of Default Template.

Assigned
: Shows whether this Template has been assigned to any specific menu items. To assign a template to menu items, open the Template for editing.

Version
: The version number of the Extension.

Date
: The date this extension was released.

Author
: The author of this extension.

Display #
: The number of items to display on one page. If there are more items than this number, you can use the page navigation buttons (Start, Prev, Next, End, and page numbers) to navigate between pages. Note that if you have a large number of items, it may be helpful to use the Filter options, located above the column headings, to limit which items display (*where applicable*).

Toolbars

At the top right you will see the toolbar:

Default

Select the Template that you want to be the default Template. Then click this button. The default star symbol will show in the Default column, indicating that this is now the default Template.

Edit

Select one item and click on this button to open it in edit mode. If you have more than one item selected (where applicable), the first item will be opened. You can also open an item for editing by clicking on its Title or Name. See the section below called Template - Edit for information on the edit screen.

Help

Opens this Help Screen.

Plugins Manager

Overview

The Plugin Manager allows you to enable and disable Joomla! Plugins and to edit a Plugin's Details and Parameters.

The Plugin Manager can be accessed by selecting "Extensions" > "Plugin Manager" from the drop-down menu on the Back-end of your HUBzero installation.

Column Headers

An indexing number automatically assigned by Joomla! for ease of reference.

Checkbox

Check this box to select one or more items. To select all items, check the box in the column heading. After one or more boxes are checked, click a toolbar button to take an action on the selected item or items. Many toolbar actions, such as Publish and Unpublish, can work with multiple items. Others, such as Edit, only work on one item at a time. If multiple items are checked and you press Edit, the first item will be opened for editing.

Plugin Name

The Name of the Plugin. Click on the Name to open the Plugin for editing.

Enabled

A green tick or a red X showing whether the use of the component is enabled/disabled. Click the icon to toggle the item between enabled and disabled.

Order

The order to display items. If the list is sorted by this column, you can change the order by clicking the arrows or by entering the sequential order and clicking 'Save Order'. Note that the display order on a page is set in the Parameters - Advanced section for each Menu Item. If that order is set to use something other than 'Order' (for example, 'Title - Alphabetical'), then the order value in this screen will be ignored. If the Menu Item Order parameter is set to use 'Order', then the items will display on the page based on the order in this screen.

Access Level

Who has access to this item. Current options are:

- Public: Everyone has access
- Registered: Only registered users have access
- Special: Only users with author status or higher have access

You can change an item's Access Level by clicking on the icon in the column.

Type.

The Type of the Plugin. Possible types are: authentication, content, editors, editors-xtd, search, system, user, and xmlrpc. These are also the names of the subfolders where the Plugin files are located. For example, Plugins with a Type of "authentication" are located in the folder "plugins/authentication".

File.

The name of the Plugin files. Each Plugin has two files, a ".php" file and a ".xml" file. So, for example, the Authentication - Joomla! plugin has two files: "joomla.php" and "joomla.xml".

ID

The ID number. This is a unique identification number for this item assigned automatically by Joomla!. It is used to identify the item internally, for example in internal links. You can not change this number.

Display #

The number of items to display on one page. If there are more items than this number, you can use the page navigation buttons (Start, Prev, Next, End, and page numbers) to navigate between pages. Note that if you have a large number of items, it may be

helpful to use the Filter options, located above the column headings, to limit which items display (*where applicable*).

Toolbars

At the top right you will see the toolbar:

Enable
: To enable one or more items, select them using the Checkbox and press this button. You may also toggle between Enabled and Disabled by clicking on the icon in the "Enabled" column.

Disable
: To disable one or more items, select them using the Checkbox and press this button. You may also toggle between Enabled and Disabled by clicking on the icon in the "Enabled" column.

Edit
: Select one item and click on this button to open it in edit mode. If you have more than one item selected (where applicable), the first item will be opened. You can also open an item for editing by clicking on its Title or Name.

Help
: Opens this Help Screen.

List Filters

You can filter the list of items by typing in part of the Title or the ID number. Or you can select a combination of Category and Published State.

Filter

In the upper left corner just above the column headings is a Filter field and two buttons, as shown below:

If you have a large number of items on the list, you can use this filter to find the desired item(s) quickly. Enter either part of the title or an ID number and press 'Go' to display the matching items. You can enter in whole words or part of a word. For example, "ooml" will match all titles with the word "Joomla!" in them.

Filter by Type and State

In the upper right area, above the column headings, are 2 drop-down list boxes as shown below:

The selections may be combined. Only items matching both selections will display in the list.

Select Type
: Select a Type from the drop-down list box to select only Plugins of this Type.

Select State
: Select a state (Enabled or Disabled) from the drop-down list box to select only Plugins with this state.

Edit Plugins

You can edit details and parameters for Plugins. Some Plugins have several parameters, while others don't have any.

Details

The Details section is the same for all Plugins, as follows:

Name
: The Name of the Plugin.

Enabled
: Whether or not this Plugin is enabled.

Type
: The Type of the Plugin. This value cannot be changed.

Plugin File
: The name of the Plugin file. Each Plugin has two files with this name. One has the file extension ".php" and the other has the file extension ".xml".

Access Level
: Who has access to this item. Current options are:

  - Public: Everyone has access
  - Registered: Only registered users have access
  - Special: Only users with author status or higher have access

  Enter the desired level using the drop-down list box.

Order.
: he order this item will display in the Manager screen. Use the drop-down list box to change the Order. You can select 'First' or 'Last' to make this the first or last item. Or you can select an item from the list. In this case, the current item will be listed just *after* the item you select. Note that the Order can also be changed in the Manager screen.

Description.
: The description of what this Plugin does. This cannot be changed.

Parameters

This will vary between plugins.

Uninstalling

Overview

See Uninstalling Extensions in the Web Developer's Guide for details.

Articles

Overview

Let's start with some definitions. In Joomla!, an Article is some written information that you want to display on your site. It normally contains some text and can contain pictures and other types of content. For many Joomla! sites, articles form the majority of the information presented in the website.

It is important to understand that, with Joomla!, the content of the site (for example, Articles) is totally separate from the formatting of the site — the way it looks on the page. So it is best to think of articles as pure content, independent of the way it might be presented. For example, the same Article might be shown with different fonts, colors, headings, and background, and might be shown in different locations on the page.

Sections & Categories

Overview

Sections and categories in Joomla! provide an optional method for organizing your articles. Here's how it works. A Section contains one or more categories, and each Category can have articles assigned to it. One Article can only be in one Category and Section.

For example, you might have a Section called "Pets", and categories called "Dogs" and "Cats". Articles about dogs would be assigned to the "Dogs" Category, ones about cats the "Cats" Category. So an Article about dogs would be in the "Pets" Section and the "Dogs" Category. You could not have one Article that is in both the "Cats" and "Dogs" categories. To work around this, you could either (1) create a new Category called "Cats and Dogs" or (2) create a second Article to put in the second Category.

Why Use Sections & Categories?

There are two main reasons you might want to organize your Articles in sections and categories.

List and Blog Layouts

First, there are built-in Menu Item Types in Joomla! that take advantage of this organization. These are the Section Blog, Section List, Category Blog, and Category List. These Menu Item Types (also called "layouts") make it very easy to show articles that belong to sections or categories. As new articles are created and assigned to sections and categories, they will automatically be placed on these pages according to the parameters you set for each page.

For example, say you have a Section Blog layout for the "Pets" section, and say you have it set to order articles starting with the most recent one first. When you add a new Article to the "Pets" Section, it will automatically show on the "Pets" blog page as the first Article. You don't have to do anything other than add the Article and assign it to the "Pets" Section.

Organizing Articles in Article Manager

If you will have a large number of articles on your site, a second reason to use sections and categories is to simply group the articles so you can find them. For example, in the Article Manager, you can filter articles based on Section or Category. So if you have 200 articles in your site, you can more easily find an Article if you know it's Section or Category.

Sections and Categories vs. Menu Organization

It is important to understand that, just because Joomla! uses a 3-tier organization level for

articles (Section ? Category ? Article), this has nothing to do with the structure of the menus on your site. For example, your site could have one menu level or six menu levels.

Other Types of Categories

There is also a potential for confusion about categories. Sections are only used for articles. However, categories are also used for other components, including Banners, Contacts, News Feeds, and Weblinks. These categories are completely different from Article categories and are set up in different screens in the back end of Joomla!. So when you see something about categories, it might refer to Article categories or it might refer to categories for these other components.

Creating a Section & Category Hierarchy

Coming soon.

Creating/Editing

Creating an Article

1. First login to the administrative back-end.
2. Once logged in, find “Content” in the main menu bar located toward the top of the page. You should be presented with a drop-down menu containing items such as “Article Manager”, “Article Trash”, etc.

3. Choose “Article Manager” from the available options.

4. Then click the "New" toolbar button.

5. The New Article screen contains options for categorizing and naming the article, editing content and selecting parameters.

    - Enter a title in the "Title" field. This is used when the article title is displayed.
    - Enter an alias in the "Alias" field. The alias is used to refer to the title and is important for the URL of the page. If you do not enter anything Joomla! will generate it for you.
    - Select a "Section" and "Category" using the drop down menus.
    - Choose whether the article is published or not using the "Published" radio buttons.
    - Choose whether the article will be displayed by the Frontpage component using the "Front Page" radio buttons.

6. You may also choose Parameters for the article. Click on each section to view the parameters and change the settings to suit your requirements.

7. Click the "Save" or "Apply" toolbar button to save your article.

    - The "Save" toolbar button will save your changes and return you to the Article Manager screen.
    - The "Apply" button will save your changes but leave you in the Article Edit screen.

Editing an Article

1. First login to the administrative back-end.
2. Once logged in, find “Content” in the main menu bar located toward the top of the page. You should be presented with a drop-down menu containing items such as “Article Manager”, “Article Trash”, etc.

3. Choose “Article Manager” from the available options.

4. You should now be presented with a list of all the content articles on your site. There are a variety of methods to find the specific entry you wish to edit: you can filter by selecting section, category, author, or even state (published, unpublished, archive). The about page should be in the section “about” and category “about”. You may also search for “About” in the filter search box or scroll to the bottom of the page and navigate your way through the entire list. Once found, click the article title to edit it.

5. You may then edit the page title, content, etc.

6. Once you feel ready to save your changes, scroll back to the top of the page and click “Save” (the icon that looks like a floppy disk) in the upper right portion of the page.

Adding an Image

Images are added to articles using the *Editor Buttons* below the content editor window in the Edit Article screen. Note: It is possible to insert images using the editor in Joomla! however this feature provides a simple way of inserting images stored in the *images/stories* directory of Joomla!

1. Open the Article for editing either by:
    - Click the **Content > Article Manager** menu item to go to the *Article Manager*, select the Article and click the *Edit* toolbar button.
    - Clicking the *Add New Article* button in the Control Panel.

    - If logged in to the Front-end, you have appropriate permissions and are viewing the Article you wish to edit: Click the *Edit* toolbar button.
2. Choose where you would like your image placed in the flow of the text by moving the cursor and click the Image editor button at the bottom of the screen.
3. The Insert Image screen will open over the top of the Edit Article screen. Choose an image by clicking on it.
    - You may also move directory by clicking on the folder icons.
    - Use the drop down *Directory* field to quickly select a directory and click the *Up* button to go up a directory level.
4. Set the image properties as required:
    - **Image Description**: This becomes the **alt** attribute for the image, an important feature for accessibility and compliance with web standards.
    - **Image Title**: Used for the optional caption and also becomes the **title** attribute in HTML.
    - **Align**: Sets the image alignment. If left blank the align attribute is not set.
    - **Caption**: Enables the caption which displays the Image Title below the image.
5. Click the *Insert* button to insert the image. The Insert Image screen will close and the image will be displayed in the editor.
    - Click the *Cancel* button to leave the Insert Image screen.

Uploading images using the Insert Image screen

You may also upload new images using the Upload section of the Insert Image screen.

1. Click the Browse button to open a file browser.
2. Select the image files you wish to upload. Click Open in the file browser to confirm the selection. Note: The file browser style and layout depends on the browser and operating system you are using.
    - To remove selected files before uploading click the red icon next to the listed files.
    - If you are uploading multiple files in batches you can clear the list of completed files by clicking the *Clear Completed* button.
3. The selected file(s) appear as a list at the bottom of the Insert Image screen. Click *Start Upload* to begin uploading files.
    - When the upload is complete a green tick will appear next to the file.
4. You may now select and insert the uploaded image as before.

Metadata

Overview

Metadata are data about data, for instance a description of a website. Metadata are important for search engines and are written in HTML. Exactly how important metadata are in the rating by search engines is debatable. However, metadata represent a good way to describe your website in short and precise words.

Note: Google does not use the keywords meta tag in web ranking. See the following article for details.

Available Tags

If you check out the HTML source code of a Joomla! site, you will see the following meta tags in the top area:

```
<meta name="generator" content="Joomla! 1.5" />
<meta name="description" content="Joomla! - the dynamic portal engine
and content management system" />
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta name="robots" content="index, follow" />
<meta name="keywords" content="joomla, Joomla" />
```

Of these, the two attributes description and keywords can be set on an article-by-article basis.

description
: This description of site content is often displayed by search engines. You should pay special attention to this tag, since the user who has entered a search term often uses this information to make a decision of whether to visit a site or not. You can add further information about every individual page to the global description.

keywords
: Keywords are very important for a document. You should list the most important concepts of your website with them. Some search engines index keywords in particular. The individual words are separated by commas; several words can be entered between two commas with normal spaces in between. The keywords should be limited to a maximum of 1000 characters; more than that will not be read by the search engines. Note that the use of fewer keywords helps each individual word get a higher priority in the search engine. Deliberate carefully about which are the most often used keywords and which are likely to be searched for most. You can add more keywords to the global keywords for every individual page of your website.

Each article may also specify, in its parameters, whether to include a title or author attribute.

Show Title Meta Tag
: This parameter will insert the following meta tag with the title of the article:

```
<meta name="title" content="Welcome to Joomla!" />
```

Show Author Meta Tag
: This parameter will insert the following meta tag with the author of the article:

```
<meta name="author" content="Administrator" />
```

The Home Page

Setting a Home Page

By default in Joomla! 1.5.x the first page in the main menu is also the default home page for your site.

If you wish to change the current home page to a different article, you must add the desired article to the main menu and make it the first item.

Edit

First login to the administrative back-end.

Once logged in, find “Content” in the main menu bar located toward the top of the page. You should be presented with a drop-down menu containing items such as “Article Manager”, “Article Trash”, etc.

Choose “Article Manager” from the available options.

You should now be presented with a list of all the content articles on your site. There are a variety of methods to find the specific entry you wish to edit: you can filter by selecting section, category, author, or even state (published, unpublished, archive). The home page should **not** be in a section or category (uncategorized). You may also search for the title of your home page in the filter search box or navigate your way through the entire list. Once found, click the article title to edit it.

The home page may look a little different than other pages you've edited, such as the "About" page. Some of the content you see on the front page doesn't appear to be in the page you're editing! This is because the home page embeds modules to output some of its content. This is done with {xhub} tags:

```
{xhub:module position="frontLeft" style="-1"}
```

You may assign multiple modules to a position. The {xhub} tag will output **all** active (published) modules assigned to the position specified. The style attribute determines if the module(s) should be wrapped in a containing div or not and is optional.

Note: The animated banner and welcome message on the front page are also modules and managed through Joomla's "Module Manager" found under the "Extensions" menu item.

Once you feel ready to save your changes, scroll back to the top of the page and click "Save" (the icon that looks like a floppy disk) in the upper right portion of the page.

Banner

XFlash

The rotating flash banner is designed to highlight important announcements, recent or upcoming events, new website features or other information valuable to your hub community. The banner is commonly placed as a central feature on a hub's homepage.

You can change the dynamic content of the banner through administrative interface. To do so:

First login to the administrative back-end.

Once logged in, find “Components” in the menu bar located toward the top of the page. You should be presented with a drop-down menu containing items such as “Module Manager”, “Plugin Manager”, etc.

Choose “XFlash” at the bottom of the list.

You will be presented with a form to fill in titles, subtitles, brief descriptions, images and label/ URL to a page with additional information for each slide. To ensure best possible display, the maximum number of slides allowed is limited to 9. You have an option of displaying text as dynamic Flash text fields, by filling in titles etc., or you can make text part of the background image. Integrating text as part of the background image gives you the flexibility to move text around within the standard 600×230 px space. To design your images, you can use a Photoshop template that we can provide you with.

For each slide, you can specify its type: regular, quote or resource. Choose “general” for slides with general announcements. To feature quotes received from users via the hub success story form, select type “Quote”. This will only work if you have quotes available, otherwise the slide will be skipped from rotation. You can also feature resources from your hub, and these will be randomly rotated within the slide with type “Resource”. To enter information about featured resources, use the Featured area at the very bottom of the form. You will need a resource id or alias name for Flash to compile correct URL. By going to Parameters of the XFlash component at the top of the screen, you can specify how many resources you want to feature. This is one good way to include more content within the 9 rotation slides.

If you use images within your banner, either as backgrounds or embedded images, you can upload them through a designated image upload area, and then reference them in the slide edit form by providing the file name, e.g. somefile.jpg. You do not need to specify a directory name, since the upload form will automatically place images in the right spot.

Slideshow (Flash)

--

Sliding Panes (Javascript)

--

Welcome

The “welcome” text, typically found to the right of the Flash banner on the home page is a module. To edit this module (or any module):

First login to the administrative back-end.

Once logged in, find “Extensions” in the main menu bar located toward the top of the page. You should be presented with a drop-down menu containing items such as “Module Manager”, “Plugin Manager”, etc.

Choose “Module Manager” from the available options.

You should now be presented with a list of all the modules installed on your site. There are a variety of methods to find the specific module you wish to edit: you can filter by selecting position, type, or even state (enabled, disabled). The welcome module would be in position “welcome” and of type “mod_custom”. You may also search for “welcome” in the filter search box or scroll to the bottom of the page and navigate your way through the entire list. Since modules are alphabetically sorted by their position by default, “welcome” will most likely be on the last page of results. Once found, click the module name to edit it.

Once in edit mode you will be presented with some options such as the title, position, etc. To edit the actual content of the module, scroll down until you see a large text area called “Custom Output”.

Edit the HTML/text to be outputted.

Scroll back to the top of the page and click “Save” (the icon that looks like a floppy disk) in the upper right portion of the page.

Your changes should now be applied and immediately viewable on the front page.

Deleting

Moving To The Trash

1. Login to the Joomla! Administrator Back-end.
2. Go to "Content" ? "Article Manger" on the toolbar menu.

3. Check the box next to the article you want to trash.

4. Then click "Trash" button. This will move the article to a trash bin so that it can be retrieved later if needed. It does not permanently delete the article.

Permanently Deleting

1. If you want to make sure the Article is completely gone from the system (so no further roll back is available), go to "Content" ? "Article Trash" on the toolbar menu.

2. Check the box next to the article you want to delete.

3. Then click "Delete" button. This will permanently delete the article. Recovery is not possible after this action.

Note: Articles in the Trash Manager are not the same as Archived Articles.

Archiving

Archiving an Article

Archiving is useful if you publish a lot of articles and which you still want to be published on the site, but have been displaced in priority by new ones.

1. Login to the Joomla! Administrator Back-end.
2. Go to "Content" ? "Article Manger" on the toolbar menu.

3. Check the box next to the article you want to archive.

4. Then click "Archive" button.

Unarchiving an Article

1. Login to the Joomla! Administrator Back-end.
2. Go to "Content" ? "Article Manger" on the toolbar menu.

3. Check the box next to the article you want to unarchive.

4. Then click "Unarchive" button.

0.8.0

Content Modifiers

Loading Modules in Articles

Modules may be loaded in an article by including a specific {xhub:module} tag. This tag requires one attribute: position, which specifies the position that you wish to load. Any modules assigned to the specified position (set via the administrative Module Manager) declared in the position attribute will have their output placed in the article in the location of the {xhub:module} tag.

```
{xhub:module position="footer"}
```

Note: To use this feature, the xHUB Tags plugin for content must be installed and active.

Including CSS or JavaScript

Sometimes we need to load a stylesheet or script for a specific article. HUBzero allows this through the use of the xHUB content plugin. CSS and JavaScript may be loaded in an article by including a specific {xhub:include} tag. This tag has a few attributes, each with multiple options:

type
: Required. Determines what sort of file to be included. Accepts "script" or "stylesheet".

filename
: Required. The name of the file to be included.

component
: Optional. The name of the component the file is to be included from. If not specified, the file will be included from the template's directory.

```
<!-- A stylesheet called "home.css" from the current template's /css f
older -->
{xhub:include type="stylesheet" filename="home.css"}

<!-- A script called "resources.js" from the com_resources component -
->
{xhub:include type="script" filename="resources.js" component="com_res
ources"}
```

Note: To use this feature, the xHUB Tags plugin for content must be installed and active.

Outputting Hub Configurations

Sometimes we need to output the value of a specific hub configuration setting. HUBzero allows this through the use of the xHUB content plugin. HUBzero based configurations (**not** Joomla!) may be loaded in an article by including a specific {xhub:getCfg} tag. This tag has one attribute: the name of whatever configuration variable you wish to output.

```
{xhub:getCfg hubShortName}
```

Note: To use this feature, the xHUB Tags plugin for content must be installed and active.

We also advise extreme caution when using this particular {xhub} tag.

See Hub Configuration for more information on what configurations are available and how to set them.

Getting The Template Directory

Sometimes we need to include content (images, javascript, etc.) from the active template's directory. HUBzero allows this through the use of the xHUB content plugin. The name of the active template will be outputted in an article by including the templatedir xhub tag. This tag has no attributes.

```
<img src="{xhub:templatedir}/images/logo.png" alt="Our Logo" />
<!-- outputs: -->
<img src="/templates/mytemplate/images/logo.png" alt="Our Logo" />
```

Note: To use this feature, the xHUB Tags plugin for content must be installed and active.

URLs

Understanding URLs for Sections, Categories, and Articles

When SEF URLs are being used, it can become tricky at times to discover which article a URL path leads to. HUBs try to simplify some of this and make the association(s) of the path to sections, categories, and articles a little more obvious.

For the purposes of this explanation, we assume every section, category, and article has an alias. In its simplest form, a URL will map to a path of http://yourhub.org/{SectionAlias}/{CategoryAlias}/{ArticleAlias}. So, a URL of http://yourhub.org/about/this/site would map to a section with an alias of "about", a category with an alias of "this", and an article with an alias of "site".

The resulting URL will start to collapse, or shorten, when sections, categories, and articles have the same alias. For instance, if you have a section with alias "about", a category with an alias of "this", and an article with an alias of "this"– notice it is the same as the category–the resulting URL will be http://yourhub.org/about/this.

If all three portions have the same alias, the http://yourhub.org/about/this/site shortens further. A section alias of "about", category alias of "about" and article alias of "about" will simply produce http://yourhub.org/about.

Quick Reference	Section	Category	Article	URL
	about	this	site	http://yourhub.c /this/site
	about	this	this	http://yourhub.c /this
	this	this	site	http://yourhub.c ite
	about	this	about	http://yourhub.c /this/about
	about	about	about	http://yourhub.c

URL Redirects

If you find yourself wanting to redirect a certain URL to another page, then *Joomla!* has a solution for you. And fortunately, the solution is as simple as creating a menu.

1. Navigate to "Menus -> Menu Manager" on the Joomla! backend

2. At this point, you can either create a new Menu or use an existing one. If you prefer the former, click "New" in the upper right-hand corner, and give your new Menu a name such as "Redirects". Alternatively, you could use the "Default" menu. The only thing to remember is that you should use a menu that is not displayed on the site (e.g. the "Main Menu" and "About" menus or displayed on the standard hub)
3. Irrelevant of which option you chose, select the icon in the "Menu Item(s)" row to add a new Menu Item to your Menu to handle the redirect
4. Next, fill in the necessary fields
    - Add a "Title"
    - The "Alias" will be the address that a user would enter into the address bar of their browser
    - The "Link" will be the page where the actual content exists

Example: According to the image below, if a user types http://myhub.org/example, though that page does not exist, they will get the content found at http://myhub.org/example/examplearticle1

5. Select "Save"
6. That's about it. Try typing the correct address into your browser to make sure that it works

Menus

Organizing Your Articles Using Menus

If you have a set of articles that you would like to group together, you can use *Joomla!* menus to organize those articles in an easy-to-navigate fashion.

An example of this can be found on the default hub under the About (yourhub.org/about) section (see also below).

1. To begin, find the URLs of the articles that you're interested in grouping together

Note: If you can't find, or don't know the URL, it comes from the article alias, the section and the category (see below and view these documentation pages: Sections and Categories, URLs).

2. With your article pages created, you'll need to make the menu structure
3. Navigate to "Menus -> Menu Manager" on the *Joomla!* backend and select "New"

4. Give your new menu a unique name and title, then click "Save"

Now you will see your new menu in the list.

5. Click the icon indicated below to start adding menu items to your new menu

Note: It is important to realize, for clarity's sake, that there is a different between "Menu" and "Menu Item". "Menu Item" represents an instance, or individual menu link, while "Menu" represents the entity as a whole.

6. Now click the "New" button to add new menu items to your menu

7. After clicking "New", scroll down and select "External Link"

8. Next, fill in the necessary information
    - This is where you'll need to remember the URLs for the articles that you've created (you'll add the URL to the "Link" field)
    - After creating this menu item, you'll be able to access this article page via the URL for the article, or the Alias of the menu item

For example: according the screenshot below, both http://yourhub.org/example/examplearticle1 & http://yourhub.org/article1 will work when trying to access this page via the browser (See URL Redirects for more details). For the navigational menus to display properly, though, you'll want to use the menu item alias when trying to access your article page (http://yourhub.org/article1 in this example).

9. Click "Save" in the upper right-hand corner

If you switch over to the front-end and navigate to your article page (via the menu item URL), you should see a page similar to this one:

10. At this point, simply repeat the previous steps to create menu items for all of the articles that you want to use together

The final step in this process is to assign the menu items that you've created to display on the article pages.

11. To do this, navigate to "Extensions -> Module Manager" on the backend *Joomla!* interface
12. Next, select "New"

13. Scroll down and select "Menu" from the list (either click on the link, or select the radio button then scroll back up and click "Next")

14. Now fill in the correct information...make sure to:
    - Give it a "Title"
    - Be sure "Enabled" is set to "Yes"
    - Give it the position "Left"
    - Set it to display on the correct pages
        - Under menu assignment, "control click" the menus items that we made in the preceding steps. This is what tells *Joomla!* which pages to display our new menu on.
    - Under "Module Parameters", "Menu Name" should be the title of the menu that we created earlier ("example" in this case)
    - "Menu Style" should be "List"

15. When all of this has been done, click "Save"
16. Now if you navigate back to your article page on the front-end, you should find something similar to this:

Members

Overview

By default casual visitors are allowed to register themselves on your website to gain access to additional resources. For example, registered users might be allowed to submit resources or simulation tools. Your Joomla! website implements a user registration policy which can be adapted to suit your particular requirements so you decide what additional resources are made available. You can disable user registration altogether if you prefer.

Users register themselves on your website using the "Register" link typically found in the upper right-hand portion of every page (position may vary). Once registered, users will log in to your website by entering their username and password using the login form found at http://yourhub.org/login. There are also links on the form to manage users who forget their usernames or passwords.

See Configuring Registration to learn more about changing registration settings.

User Manager vs Members Component

User Manager

Joomla! comes with a User Manager accessed via the back-end from the "Site" drop-down of the main menu.

Members Component

HUBzero extends the standard Joomla! user profile with options for more demographic information (race, sex, etc.) and various other key pieces of information needed for such things as simulation tools. All this information can be accessed under "Members" in the "Components" list found in the main menu of the back-end administration.

When to use which

All user will have an entry in **both** the User Manager and the Members Component. Any information edited with one, such as changing a user's name via the User Manager, will also be automatically reflected in the other. Since the majority of fields in the User Manager are also represented in the Members Component plus considerably more fields, it is recommended that all user editing be done in the Members Component. The one exception concerns adjusting a user's access level (regular user vs. administrator, etc.). This is still done only with the User Manager.

Creating by Proxy

Proxy-creating users

To create an account for another user:

Note: This feature is available **only** to logged-in site administrators.

1. Log in on the hub's front-end interface.

2. Once logged in, navigate to **YOURHUB.org/register/proxycreate** (for hubs with older code, use: **YOURHUB.org/registration/proxycreate**). Most likely you will have to type this URL directly into your browser as it is generally not linked to anywhere in the hub content.

3. You will be presented with a form asking for information about the user you are creating (specifically, fields that were set to be visible under the Proxy column in the "HUB Configuration: Registration" settings). Fill in all required fields.

   Note: Usernames cannot be changed, but passwords may be changed anytime after account creation.

4. Click **Proxy Create Account**.

5. When the account is created successfully, you will be shown the basic text of an email which you **MUST** then copy and paste and send to that person. This email will provide them the username and initial password you have set for them and links they could use to confirm their email and change their password. You may add any other information about contributed resources or the reason for their account you deem appropriate.

Editing

Overview

Note: Editing of users is done through the HUBzero Members Manager component, **not** the standard Joomla! User manager. Any changes made via the Members Manager will also be reflected in the Joomla! User tables. Making changes with the Joomla! User Manager can lead to data becoming out of sync.

1. First login to the administrative back-end.
2. Once logged in, find “Components” in the main menu bar located toward the top of the page. You should be presented with a drop-down menu containing a list of all installed components.

3. Choose “Members” from the available options.

4. You should now be presented with a list of all the members on your site. There are a variety of methods to find the specific person you wish to edit: you can search by such fields as name, email, and ID number. You may also scroll to the bottom of the page and navigate your way through the entire list. Once found, click the person's name to edit their information.

5. You may then edit their name, organization, employment status, etc.

6. Once you feel ready to save your changes, scroll back to the top of the page and click “Save” (the icon that looks like a floppy disk) in the upper right portion of the page.

Adjusting Access Level

Joomla! offers various levels of access and privileges for users. All new accounts, by default, will be *Registered Users*, the access level with the least amount of privileges. Sometimes, you will need to grant a (trusted) user administrative access so they can manage content and portions of your hub. We strongly advise that administrative privileges be handed out rarely and with caution.

To adjust a user's access level:

1. First login to the administrative back-end.

2. Once logged in, go to the "User Manager." The User Manager can be found by selecting "Site" > "User Manager" from the drop-down menu on the back-end of your HUB installation.

3. Choose the user you wish to edit from the available list.

4. Once the page has loaded, find the “Group” option, found on the left-hand portion of the screen under the "User Details" grouping.

   The available access levels are as follows:

   - *Registered User:* Normal visitors who register at the site. Can view Menu Items that have Access Level of Registered. Cannot edit or submit articles.
   - *Author:* Can submit new articles for approval in the front end only. A Publisher or higher must approve. Cannot edit existing articles.
   - *Editor:* Can submit new articles or edit existing articles from the front end only. A Publisher or higher must approve.
   - *Publisher:* Can submit, edit, or publish articles from the front end only.
   - *Manager, Administrator and Super Administrator:* Can do all of the above plus can log into the back end with increasing rights.

5. Select the access level desired for the user and then click "Save" (the icon that looks like a floppy disk) in the upper right portion of the page. Changes take affect immediately.

0.8.0

Deleting

Overview

In order to delete a user from your database, log in to the administration panel and choose "Site" -> "User Manager" from the top menu. On the opening page, select the checkbox to the left of the user's name you would like to remove and press the "Delete" button in the upper-right corner.

Looking Up

Overview

There will be times you need to track down a user's information for whatever reason. This can be a time-consuming task when presented with paging through potentially thousands of accounts. Luckily, there are multiple ways you can search and narrow down results quickly.

User Manager

Since information such as user ID, name, username, and email is kept consistent between the User Manager and Members Component, you can use either one to search for a specific account.

To search via the User Manager:

1. First login to the administrative back-end.

2. Once logged in, go to the "User Manager." The User Manager can be found by selecting "Site" > "User Manager" from the drop-down menu on the back-end of your HUB installation.

3. You will be presented with a list of all users registered on your site.

4. Above the list of users are a few options for filtering results.

   On the left side is a search box. You can search for users by name, username, or email address.

   On the right side are a few select boxes that allow you to filter the users based on their access level (editor, administrator, etc.)—here, it's called "Group"—or log status (logged in/out).

5. Choose your method for filtering and the list of users will be narrowed down to match those criteria. Once found, click the person's name to view/edit their information.

Note: If you need to find an account to access more information than name, username, and email, it is recommended to search via the Members component.

Members Component

From the Administrative Back-end

Since information such as user ID, name, username, and email is kept consistent between the User Manager and Members Component, you can use either one to search for a specific account. The Members Component, however, will allow access to more details about a user's account. The extended demographic information provided by the Members Component, for example, cannot be accessed via the User Manager.

To search via the Members Component:

1. First login to the administrative back-end.
2. Once logged in, find “Components” in the main menu bar located toward the top of the page. You should be presented with a drop-down menu containing a list of all installed components.

3. Choose “Members” from the available options.

4. You should now be presented with a list of all the members on your site. There are a variety of methods to find the specific person you wish to edit: you can search by such fields as name, email, and ID number.

5. On the left side, above the list of members, is a search field for filtering the list. You can search by such fields as name, email, and ID number.

6. Choose your method for filtering and the list of users will be narrowed down to match those criteria. Once found, click the person's name to view/edit their information.

From the front-end

The Members Component offers a "whois" feature on the front-end that allows for more complicated queries than the back-end.

Note: This feature is available **only** to logged-in site administrators.

1. First login to the front-end.
2. Once logged in, navigate to "/members/whois". Most likely you will have to type this URL directly into your browser as it is generally not linked to anywhere in hub content.

3. You will be presented with a rather simple form containing one input box. This is where you'll build your query.

4. Fields that can be searched:

   name
   : User Name

   email
   : User Email address

   username
   : User login/username

uidNumber
: User ID (numerical)

emailConfirmed
: Status of the email confirmation process

proxyUidNumber
: User ID (numerical) of accounts created by proxy

proxyConfirmed
: Status of the email confirmation process for accounts created by proxy

Operators available:

*
: Wildcard

?
: Wildcard

&lt=
: Less than

&gt=
: Greater than

=
: Equals

!=
: Does not equal

The following example query would search for all users with a "@hubzero.org" email address:

```
email=*@hubzero.org
```

5. Submission of the form will return a list of users matching the criteria of the query. Once found, click the person's name to view/edit their information.

Points

Default Settings

Private/Public Profiles

By default, new profiles are set to "private". This means that only super administrators and the member him/herself may see and access that profile's information. Private profiles do not display in search results.

Public Profiles by Default

Member profiles may be set to public by default via the administrative Members component. To do so, log in to the administrative back-end and find "Members" in the "Components" list of the main menu. Once the Members manager page has loaded, locate the "Parameters" button in the toolbar (top right, opposite of the "Members" heading). Click this. You should be presented with a series of settings. Find "Default Privacy" and set to "Public". All new accounts will now default to public profiles.

Note: any accounts created before changing this setting will retain whatever configuration they had. For instance, private profiles will remain private unless manually changed by the user.

MyHUB Layout

To change the default MyHUB layout, first...

1. Log on to the *Joomla!* back-end interface for your hub (https://yourhub.org/administrator)
2. Navigate to "Components -> My HUB" on the main menu bar at the top of the screen
3. Click the "Customize" button located in the top-right corner of the screen

4. Add/remove/move the modules to create the default My HUB layout for new users

5. Finally, select "Save"

Note: any accounts created before changing this setting will retain whatever configuration they had.

Daily Maintenance

The Dashboard

Every HUB comes with an administrative dashboard that gives you a quick overview of some activity or items on the site that need attention.

First login to the administrative back-end.

Once logged in, you should be presented with a dashboard of categories and the activity within them, such as “Abuse Reports”, “Support Tickets”, etc. Items that require attention are highlighted in red and give direct links to their corresponding components so you may address the issue presented.

For instance, in the screenshot below, the dashboard says there are 46 unassigned support tickets. By clicking on the link “46 unassigned tickets”, you will be immediately taken to the support ticket manager. From there you may assign, update, comment on, or close tickets.

Resources

Approving Pending Resources

Depending on whether auto-approval for user-submitted resources on a hub is turned on/off, you may need to watch for newly submitted resources awaiting for administrator approval to be published on the hub. The administrative dashboard shows whether there are any resources pending approval.

In the screenshot above, the dashboard says there are 49 pending resources. By clicking on the link “49 pending resources”, you will be immediately taken to the resource manager. Resources that need your approval will be marked with a distinct icon as seen in the screenshot below.

You can publish a pending resource just by clicking on the orange status icon, or you can first click on the resources title and select appropriate status from the detailed resource page, as shown on the screenshot below.

From the detailed resource page you can also decide when the resource should get published and unpublished by setting Start Publishing and Finish Publishing dates. As soon as a new resource is published, it will appear in site search as well as in What's New area of the site.

Creating Courses

Here's how to create a "Course" Resource...

1. Begin by logging on to the administrator side of the site (http://yourhub.org/administrator/)

Here's a simplified sample of how a course is set up:

This is the structure that you will be creating.

NOTE: Before starting, it is important to include a note about naming conventions. Be as descriptive and consistent as possible. This is important because it helps to create consistency throughout your hub by applying the same naming structure to similar files. For example, a possible PDF name could be 2010.09.03-ECE565-L10.pdf (which includes date, course, and lecture). Remember, a user might download several PDF's from your course, so it helps to distinguish them appropriately.

1. Select "Resources" from the "Components" menu at the top of the screen

To start, create the highest level of the structure: the "Course"

1. Select "New" from the top right corner buttons (do not select "Add Child")

1. Enter the data for your course
    - Make sure that the "Type" field is set to Course (this is not the default!)
    - Also be sure to "Publish" the course (using the drop-down box on the right)
    - And select save

Now it's time to create the Lecture for the course (the second level)

1. Select "New" again from the top right corner buttons (again, do not "Add Child" yet)
2. Now create a Lecture

    NOTE: A lecture is not an actual resource type, but that is how it will be referenced in this documentation, as that is its function within a typical course

    - The type of actual resource chosen for a lecture is not important
        - This relationship is established via a parent/child designation, not by resource type
        - A good option is "Online Presentation" or "Teaching Material" for the Lecture resource type
        - But don't forget, the resource type ***is*** important for the "Course" resource
3. Type in the information for this lecture
    - Select an option for "Type", such as "Teaching Material"
    - Be sure to publish this resource as well
    - Select save

Now you should have your "Course" resource and your "Lecture" resource showing up

1. Note the resource number of your "Lecture/Teaching Material" (in this example it is 11)
2. Next, click the "[+]" button in the "Children" column and in the row of your Course Resource (i.e. the first resource that was created; which was 10 in this example)
3. Select the "Add existing - Resource ID:" radio button, and type in the resource ID of the "Lecture/Teaching Material" that was created

1. Click "Next"

You can now see that you have created a parent/child relationship between the course and lecture presentation

1. Click the "Resource Manager" text at the top of the screen to go back to viewing all resources

NOTE: Notice that you now have one child next to your Course resource (this is the lecture resource that you related it to in the previous steps)

Now, the last step is to add the Lecture material itself (this is tier three of our course structure)

1. Select the "[+]" button again, this time next to the Lecture/Teaching Material resource that you created
    - Yours might not necessarily be an "Teaching Material" if you chose to label it something different

1. This time leave the radio button on "Create New" and click "Next"

Here you add the actual file to be played for the "Lecture/Teaching Material" that you are creating

1. Type in the info about this media item being uploaded for the course
    - Select the appropriate information (for this file, a PDF of class slides is being uploaded, that's why "PDF" and "Presentations Slides" are chosen)
    - Remember the note on naming conventions mentioned at the beginning of this section

1. Now, select the file box on the right to upload the file for this Lecture
    - a) Click in the browse box and locate the file on your computer

NOTE: When uploading "Breeze" files, place all of the files associated with that video in a .zip folder. Then select that .zip folder to upload in the upload box shown below. Also select "Unpack File". Once uploaded, continue with the process below, while simply setting the main file as the "viewer.swf" file.

- b) Back in the file box, click "Create or Upload"
- c) Then select the radio button for the file that you uploaded and make sure the "With Selected" drop down box above says "Set as main file"
- d) If so, click "Apply" next to that drop down box
- e) This will add the URL to the "File/URL" box to the left

1. Now select Save

You have successfully created a three tier structure for your "Course". The first level is the "Course" that was created first. Then the "Teaching Material" was created; this was also called a Lecture. Then a file was added to that Lecture.

At this point, more lectures and files can be added to a Course. To add a different file type to a given lecture (say you want to post a video and a PDF for the same lecture), just follow steps 13 and following, and create a new Child under the appropriate lecture (simply click on the resource for your lecture, and click the "Add Child" button on the resource page for that lecture). You can also add more lectures to the course. Simply follow steps 5 and on, creating a new resource and then adding that existing resource as a child of the main Course resource as was done in this example. The reason resources are created in two different ways in this example is due to how they are set up. When clicking the "[+]", a child type resource is created. When the "New" button is used from the main resource page in Joomla, a parent type resource is created. The course and lecture are parent type resources (evening though you add the lecture as a child to the course), and the material is the child type resource.

Just remember, the structure/hierarchy is very important for this to work properly. Pay careful attention to maintain the structure outlined above.

Note concerning "Series": The theory outline above holds true for creating a series. A series is simply a two level structure (as opposed to three), where the series is created in the same way a lecture was, and the series resources are the children.

Tools

Contribtool

Contribtool is the process that manages the publishing process for tool resources on the hub. There are nine states of a tool within Contribtool:

Each state is associated with a set of tasks that either the tool developer or the hub administrator must perform. We will refer to tasks the tool developer must perform as "user tasks", and tasks the hub administrator must perform as "admin tasks".

Preparation

To accomplish many of the admin tasks, the user acting as the admin must be a part of the "apps" group. To be added to the "apps" group:

1. Login to the hub's administrative back end and find the "Groups" component.

2. Type "apps" into the search box.
3. Click on the number under the column "Total Members"

4. Type the admin's username into the Add input box, choose "Members" from the drop down menu, and push the "Add users" button.

Registering a Project

0.8.0

The first step in Contribtool is for the tool developer to register the tool by filling out the tool contribution form. This is a user task. Go to the web page https://yourhub.org/contribute (substituting "yourhub.org" with the name of your hub). Use the "Getting Started" button in the upper right side of the web page to start the contribution process. On the left side of the web page, a list of contribution types will be shown. Choose the TOOLS contribution type.

The tool registration form will be presented. The form asks for some basic information regarding the name of the tool, a short description of the tool, and who can access the source code repository. There are also fields available to add restrictions on who can run the tool once it has been published, and who can access the Trac project area. All of the information on this page can be changed at a later date, except for the tool name. Once the tool name has been registered, it cannot be changed, so pick a good one. When the registration form has been completed, push the "Register Tool" button at the bottom of the page, and the state of the tool will be changed to "Registered"

Registered to Created

Once the tool has been registered by the tool developer, it is up to the administrator to create the project. Creating the project is an admin task. Start by going to the web page https://yourhub.org/contribtool. This page lists out all tool contributions on the hub. Tools that require administrator attention are highlighted. If a tool has been registered by a tool developer, but there are no tool contributions listed on the Contribtool web page, there may be a privileges related error. Make sure the administrator account being used is a member of the "apps" group. Instructions on how to do this are in the section labeled "Preparation".

The newly registered project should be highlighted on the Contribtool web page. Choose this project by clicking on the tool name, which is a link to it's tool status page. The tool status page contains the tool information that the developer provided on the registration form and a set of Developer Tools on the left side of the web page. On the right side of the web page, the "What's next?" status is provided, which gives information regarding the steps that need to be completed before the tool can be published on the hub. Any tasks that have been checked off have been completed. Tasks that have an arrow next to them are tasks that can be completed now.

Special for the administrator, there is a section of "Administrator Controls" on the left side of the page, under the Developer Tools. To create the tool project, the admin must press the link labeled "Add Repo". This starts a process that creates a Subversion source code repository, Trac project wiki pages, and sets up access for the tool for the users listed on the development team. When the process has completed, the results are displayed for the administrator to see. A green box generally means nothing catastrophic has happened and the tool project are has been successfully created. In the case something fails, a red box will be displayed listing the errors encountered. Running the process multiple times by repeated presses of the "Add Repo" link generally has no harmful effects, in that if the pieces of the project area already exist, they will not be overwritten, and if they do not exist, they will be created.

The last step to creating the tool project area is to flip the status of the project from Registered to Created and pressing the "Apply change" button at the bottom of the Administrator controls.

Created to Uploaded

After the tool project has been created, the tool developer needs to upload their tool source code into the source code repository. This is a user task. Instruction listing out the commands involved with upload source code to the project's source code repository are easily accessible from the tool status page in Contribtool. Start by accessing the Contribtool web page https://yourhub.org/contribtool. Find the tool project on the Contribtool web page, and click on the tool project's name. This is a link which leads to the tool status page. While in the Created state, the "What's next?" section on the right side of the web page contains a link to the instructions. We will review the requirements for installing a tool in the HUB environment and the process of doing so.

There are four requirement for installing a tool in the HUB environment:

1. Source code
2. Graphical User Interface
3. Makefile
4. Invoke script

Source code includes all files related to running the application with the tool.xml and possibly a wrapper script as the exceptions. Source code is needed as a part of the installation process. No binaries from the source code of the application should be stored in source code repository. It is alright to store binary data files, like images, in the source code repository, but in most other cases, storing binaries from the application instead of the actual source code leads to future compatibility problems, even when the binaries claim to be platform independent.

All tools need a graphical user interface. There are several toolkits available including Qt, GTK, wxWidgets, and Tcl/Tk. If your tool does not already have a graphical user interface, we suggest using Rappture (http://rappture.org). With Rappture, you can quickly develop a graphical user interface that can guide users through the process of running the tool and viewing results, all in one application. Rappture also include many hooks into the HUB infrastructure, which makes it a great choice for tool developers.

All tools need a Makefile. The Makefile holds the instructions for compiling and installing binary files. Even tools that do not have source code that needs to be compiled still need a Makefile.

All tools need an invoke script. Invoke scripts hold details about how to launch to the tool in the HUB environment. Generally, invoke scripts are very simple, one line calls to the HUB invoke script with a few options.

While generating the source code and graphical user interface are outside of the scope of this tutorial, details will be given later regarding creation of a typical Makefile and invoke script.

There are eight steps related to uploading source code to the source code repository.

1. Checkout a copy of the tool's source code repository.
2. Add source code to the src directory of the tool's repository.
3. Add a Makefile in the src directory of the tool's repository.
4. Add tool.xml to the Rappture directory of the tool's repository.
5. Check the middleware/invoke script.
6. Test the code in the workspace.
7. Clean the directories before committing.
8. Commit the changes from the local copy of the repository.

All of these steps can be performed from within the workspace.

The first step of uploading source code to the source code repository is to checkout a copy of the tool's source code repository. This can be done by using the svn checkout command.

```
svn checkout https://yourhub.org/tools/toolname/svn/trunk toolname
```

In the example, replace "yourhub.org" with the name of the hub hosting the tool's source code repository. Also, substitute the term "toolname" with the shortname of the tool. Executing this command will download a copy of the tool's source code repository from the repository server, hosted on "yourhub.org", and place the copy on the machine where the command was executed. If the command was executed inside of a workspace, the copy of the tool's repository server will be made inside of the workspace. This copy will be referred to as the local copy of the tool's source code repository. The local copy of the repository will be stored in a directory, with the name substituted for "toolname".

The second step of uploading source code to the source code repository is to add source code to the src directory of the tool's repository. This can be done with the svn add command. This step involves (1) changing directories into the local copy of the repository, here referred to as toolname, (2) copying source code files into the src directory, and (3) lastly using the svn add command to notify the local repository that files need to be added to the source code repository.

```
Line    Example:
  1       cd toolname
  2       cp /apps/rappture/examples/zoo/curve/curve.tcl src/curve.tcl
  3       svn add src/curve.tcl
```

All files will need to be added to the source code repository. They can be done one by one, as shown above, or by directory.

The third step of uploading source code to the source code repository is to add a Makefile to the src directory. The svn add command will be used to complete this task. If a Makefile does not exists, one can easily be created using a text editor. In the workspace, the gedit program is a convenient text editor. From within the toolname directory, issue the command:

```
gedit src/Makefile
```

If no Makefile exists, add the following text to the file.

```
Line    Example:
  1       all: curve.tcl
  2
  3       install: curve.tcl
  4           install --mode 0644 -D curve.tcl ../bin/
  5
  6       clean:
  7
  8       distclean: clean
  9           rm -f ../bin/curve.tcl
```

In this example, the source code is a single tcl script named curve.tcl. Since the source code does not need to be compiled in this case, the "all" target is empty. If there was source code to be compiled, line 2 would be tabbed in once and include a compile line like "gcc -o curve curve.c". The install target, starting on line 3, is responsible for installing the executables and related scripts into the ../bin directory. Executables should be placed in the bin directory. Line 4 is the action of the install target. It uses the shell program named "install" to move the files from the src directory to the bin directory, and appropriately set the permissions of the file. The clean target, on line 6, is responsible for removing files temporary files, usually left over from compilation. The distclean target, on like 8, calls the clean target from line 6, and then removes the tcl script that was placed in the bin directory by the install target. The distclean target is responsible for removing any files that were generated be install target. After running the distclean target, the local repository should be in the same state as when it was first checked out.

Once the Makefile has been created, save the file and exit the gedit program. Lastly run the svn add command on the file src/Makefile to add it to the local copy of the source code repository.

The fourth step of uploading source code to the source code repository is to add the tool.xml file to the Rappture directory. To add the tool.xml file to the local copy of the source code repository, first copy the file into the Rappture directory, then use the svn add command to notify the local copy of the repository that a file needs to be added.

```
Line    Example:
  1       cp /apps/rappture/examples/zoo/curve/tool.xml
            rappture/tool.xml
  2       svn add rappture/tool.xml
```

The fifth step of uploading source code to the source code repository is to check the middleware/invoke script. Inside of the middleware directory, there should exist a file named invoke. If the file does not exist, use gedit to create it, then add it to the repository using the svn add command, just as was done for the Makefile in step 3. The invoke script is responsible for setting up the tool environment and launching the tool when a user clicks the Launch button from the tool resource page. The default invoke script looks like this:

```
Line    Example:
  1       #!/bin/sh
  2
  3       /apps/rappture/invoke_app "$@" -t toolname
```

In the example above, the invoke script calls the HUB invoke script located at /apps/rappture/invoke_app. On older hub setups, /apps/rappture/invoke_app is a file that only supports launching rappture applications. On newer hub setups, /apps/rappture/invoke_app is a link to /apps/invoke/current/invoke_app, which is capable of launching all rappture applications and a majority of non-rappture applications.

The invoke_app script accepts a number of flagged options. Some of the more popular ones include:

```
-r  Rappture version (current, dev)
-t  tool name
-T  tool root directory
-e  environment variable to set
-p  add a path to the PATH environment variable
-C  command to launch the tool
-A  additional arguments to add to the command for launching the tool
-c  commands to start in the background before launching the tool, lik
e filexfer
```

The example above takes advantage of the -t flag to tell invoke_app that the name of the tool is "toolname". The invoke_app script contains more details about the available flags and how to use them.

The last task of this step is to ensure the middleware invoke script has execute permissions. This is accomplished by using the shell's chmod command:

```
chmod 755 middleware/invoke
```

The sixth step of uploading source code to the source code repository is to test the tool in a workspace. If the tool runs properly in a workspace when started from an invoke script, it is very likely that it will run properly when installed in the hub environment. Testing the code in the workspace also exercises the Makefile and invoke script, ensuring they are also functioning properly. Here are the commands:

```
Line    Example:
  1       cd src
  2       make all install
  3       cd ../
  4       ./middleware/invoke -T $PWD
```

In the above example, the code is compiled and installed by (1) changing directories into the src directory, and (2) issuing the command "make all install". The shell command "make" looks for a file named Makefile, and runs the targets specified as its arguments. In this case, it runs the all and install targets. Next (3) change directories back to the tool's root directory and run the middleware/invoke script, providing the option "-T $PWD". Recall from earlier, the -T option specifies the tool root directory. The $PWD environment variable hold the present working directory, which in this case also happens to be the tool's root directory. When the Rappture GUI appears, press the simulate button to make sure the tool runs correctly.

It is important that this step is successfully completed by the tool developer. This is the same sequence of events that the administrator will do when installing the tool in the HUB environment. Any failures encountered while running this step will most likely be encountered again when installing the tool in the HUB environment, so they should be resolved prior to changing the tool status to Uploaded in Contribtool.

The seventh step of uploading source code to the source code repository is to clean up the local copy of the repository before committing. This will again utilize the Makefile located in the src directory. Here are the commands:

```
Line    Example:
  1       cd src
  2       make distclean
  3       cd ../
```

Earlier the Makefile was setup with a distclean target to clean up all files generated from the installation of the tool's source file. In this example, the distclean target of Makefile is called to do the cleanup of the previous install. It is important to clean up the repository before committing to avoid placing unnecessary binary and other temporary files into the source code repository stored on the repository server.

The eighth and last step of uploading source code to the source code repository is to commit the changes from the local copy of the repository to the repository server. This is accomplished by using the command svn commit. The svn commit command accepts the --message flag along with a message. With every commit to the repository, a message should be included detailing what is being committed. This will help later when searching for specific versions of the tool in the source code repository. Commits can be made from any directory in the local copy of the repository, but will only include files and directories underneath the present working directory. For this reason it is usually convenient to commit from the local copy of the repository's root (top) directory. Here is an example commit command with a message:

```
svn commit --message "initial upload of code"
```

After issuing the svn commit command, a prompt may appear requesting a username and password. Enter the username and password credentials for the HUB hosting the tool's source code repository.

Once the tool's source code has been uploaded to the repository server, the tool's Contribtool status can be changed to Uploaded. Go to the tool's status web page in Contribtool, it can be found through the Contribtool web page:

```
https://yourhub.org/contribtool.
```

To make sure the source code was successfully committed, use the Timeline link under the Developer Tools section on the left side of the web page. The Timeline link will open a web page with the tool's project area. Note that the tool's project area may have restricted access that requires the user to login. The timeline lists out the most recent changes to the source code repository. The most recent changes for uploaded code should be listed at the top. Make sure the uploaded code is listed in the Timeline.

Back on the tool's status web page in Contribtool, there should be a "We are waiting for You" section under the "What's next?" section on the right side. When ready to change the status of the tool from Created to Uploaded, click the link labeled "My code is committed, working and ready to be installed". Clicking the link will change the status.

Uploaded to Installed

Once the tool has been uploaded by the tool developer, it can be installed in the HUB environment. This is an admin task. Navigate to the tool's status web page in Contribtool. Under the Administrator Controls section, on the left side of the web page, will be a link labeled Install. Clicking the Install link will start the installation background process which performs an svn checkout of the tool's source code repository into the /apps directory. The next step requires that the administrator login to the system, become the "apps" user, and compile the code. This can all be accomplished within a workspace.

Inside of a workspace issue the following commands in an xterm:

```
Line    Example:
  1       sudo su - apps
  2       cd /apps/toolname/dev/src
  3       make all
  4       make install
```

Become the apps user by using the shell's sudo command. If there are errors while becoming the apps user, go back to the beginning of this tutorial and ensure the login account being used is a member of the "apps" group. Change directories to the tool's installed dev directory, /apps/toolname/dev. Inside of the src directory, run the "make all" and "make install" commands to compile and install the code.

When the source code has been compiled and installed, go back to the tool's status page in Contribtool, flip the status from Uploaded to Installed, and press the Apply change button. Pressing the Apply change button will update the tool status web page.

On the updated page, a Launch tool link will be made available, on the right side of the page, in the "What's next?" section. Click the Launch tool link and verify that the tool properly launches and runs.

Installed to Updated

It is the tool developer's responsibility to test the operation of the tool and validate that it generates acceptable results. The tool developer may find an error or otherwise needs to update the code in the source code repository and have the tool reinstalled. Updating the tool is a user task. Any new changes to the tool should be committed to the source code repository. Next, the tool's status in Contribtool should be changed to the Updated state. Changing the status will notify the administrator that the tool is ready to be installed again.

Installed to Approved

When the tool developer feels the tool is working properly, the approval process can be started. Tool approval is a user task. To approve a tool, the tool information page must be created. The tool information page is the web page listing the authors of the tool, credits, publications, and screen shots. To create the tool information page. Click the "Create this page" link in the "What's next?" section on the tool status page in Contribtool.

During the process of creating the tool information page, the necessary information will be collected and formatted. At the end of the process a preview page will be provided. The tool developer will need to confirm the layout and information on the preview tool page.

The tool developer will also need to choose a license under which the project will be published. Be sure to replace the template text in the license with the appropriate information.

Finally, approve the tool by pushing the button labeled "Approve this tool". The status on the tool's status page in Contribtool will automatically be updated.

Approved to Published

To publish a tool in the Approved state, the administrator must click the Publish link in the Administrator Controls on the tool's status page in Contribtool. This is an admin task. Clicking the Publish link updates the database, exposing the tool information page on the HUB. From the tool information page, the tool can be launched using the Launch tool link. The Publish link will

run the publishing process and return the results to the administrator. Similar to the Installed state, successful completion will result in a green box with results. Failure will result in a red box with errors. After the results have been presented, flip the status from Approved to Published. and press the Apply change button.

Published to Updated

When the tool developer is ready to start testing a new release of the tool, they will commit all changes to the source code repository, and change the status on the tool's status page in Contribtool, from Published to Updated. Changing the status from Published to Updated does not unpublish of otherwise change the state of the already published tool. Changing the status will notify the administrator that the tool needs to be reinstalled. From here, the Install-Approve-Publish cycle continues.

Published to Retire

The tool developer has the option of retiring a tool. Generally, published tools are not retired, and live in perpetuity. Changing the status from Published to Retired places the tool in a state where nobody can run it from the tool information page. The tool information page will still exist, as a placeholder for the previously published resource.

Support Tickets

Abuse Reports

Store Orders

Site Notices

Overview

1. First login to the administrative back-end.
2. Once logged in, find “Extensions” in the main menu bar located toward the top of the page. You should be presented with a drop-down menu containing items such as “Module Manager”, “Plugin Manager”, etc.

3. Choose “Module Manager” from the available options.

4. You should now be presented with a list of all the modules installed on your site. There are a variety of methods to find the specific module you wish to edit: you can filter by selecting position, type, or even state (enabled, disabled). The site notice module would be in position “notices” and of type “mod_notices”. You may also search for “notices” in the filter search box or scroll to the bottom of the page and navigate your way through the entire list. Once found, click the module name to edit it.

5. Once in edit mode you will be presented with some options such as the title, position, etc. Typically, you will not need to edit anything found in the “Details” section. The items that will most often be changed are under “Parameters”. Here you can set the following:

   Start Publishing
   : This determines when you wish the site notice to *start' showing on the site.*

   Finish Publishing
   : This determines when you wish the site notice to *stop* showing on the site.

   Alert Level
   : 3 levels available: low, medium, and high. This determines the appearance of the site notice (such as eye-catching red for “high”).

   Module ID
   : A CSS ID to be applied to the module. Typically, you will not need to change this.

   Message
   : The notice you wish to display.

   Important! The “Position” must be set to “Notices”, otherwise the banner will not show up, or it will not show up where you intended it to.

6. Once your parameters are set, move down to the “Menu Assignment” portion. This section determines what pages of your site you wish the notice to appear on. This will typically be set to “All”.

Important! If the menu assignment is set to “none”, your notice will NOT appear on any page of the site, even if it is published and the timeframe is within the publishing window set in the “Parameters” section.

7. Return to the top of the page and click “Save” (the icon that looks like a floppy disk) in the upper right portion of the page.

Components

Citations

Events

Categories

Creating a category

Click “New” (green circle with a plus sign) in the upper right portion of the page.

You will be presented with a form to enter your new category. Only the field labeled “Title” is required.

When done, click “Save” (the icon that looks like a floppy disk) in the upper right portion of the page.

Editing a category

To edit a category, click the title in the category listing or check the box next to the title and click the “Edit” button in the upper right of the page.

All other steps are identical to category creation above.

Deleting categories

To delete one or more event categories, check the box next to the category you wish to delete

and hit the “Delete” button in the toolbar located at the top right of the page.

Groups

Resources

Types

Users may contribute many types or categories of resources. All HUBs come with a default set of types. They are:

- Animations
- Courses
- Downloads
- Learning Modules
- Notes
- Online Presentations
- Publications
- Series
- Teaching Materials
- Tools
- Workshops

Some types can be contributed by any user via the front-end â€œcontributeâ€• component. Some types, such as Learning Modules or Series cannot presently be contributed via the front-end due to their complex nature (more on this below).

Creating New Types

The default list of types may not completely suit your needs and you may wish to add your own, new types. This can be done from the administrative back-end of the site by following these steps:

1. Login to the administrative back-end.
2. Once logged in, find â€œComponentsâ€• in the main menu bar located toward the top of the page. You should be presented with a drop-down menu containing a list of all the installed components on your HUB.
3. Select â€œResourcesâ€• from the list.

Once the â€œResources Managerâ€• page has loaded, select â€œTypesâ€• from the sub-menu found under the â€œResources Managerâ€• title.

You should now be presented with a list of all the resource types available. There are multiple categories of types. Select â€œMain Typesâ€• from the â€œcategoriesâ€• option found directly above the list or navigate the listing until you see those designated as â€œMain Typesâ€•. These are the primary types of resources users will be able to contribute, browse, etc.

Click the title of a type to edit it or click â€œNew Typeâ€• (green circle with a plus sign in the middle) from the toolbar on the top right of the page.

Fill in the title field (what you want this type to be called) and select â€œMain Typeâ€• from the category list. You may then optionally fill in a description (longer explanation) of this type.

Decide if this resource type can be contributed from the front-end â€œcontributeâ€• component by checking/unchecking the â€œcontributableâ€• checkbox. Due to the complex structure of some resources, the front-end â€œcontributeâ€• component cannot handle the creation of some resource types. This checkbox ensures what such a resource type does not accidently show up as a type the user can self contribute.

Next, you may decide what custom fields you wish to have filled in by the user for that resource type. Options are single-line text boxes or multi-line text areas. A field may be required or not. If no custom fields are added, the resource type will appear with a default set of fields: Bio, Credits, Citations, Sponsored By, References, Publications (all multi-lined textareas)

Click â€œSaveâ€• (icon looks like a floppy disk) in the toolbar on the top right of the page.

New resource types should now be available as a resource option and changes to types should take affect immediately.

Tags

Overview

Tags are like keywords or category labels. Tags help you categorize and find content, events, and members which have something in common or similar interests. Tags can be added to groups, profiles, resources, wiki pages, and events.

To manager tags:

1. First login to the administrative back-end.
2. Once logged in, find “Components” in the main menu bar located toward the top of the page. You should be presented with a drop-down menu containing a list of your installed components.

3. Choose “Tags” from the available options.

Creating a Tag

1. Click “New” (green circle with a plus sign) in the upper right portion of the page.

2. You will be presented with a form to enter your new tag. Only the field labeled “Tag” is required.

   Admin
   : This is a flag that designates if a tag is an administrative tag. Admin tags are only seen by administrators and are typically used for tagging content with information that may not be relevant or desired to be known by front-end users.

   Tag
   : The tag you wish to create.

   Alias
   : An alternate version of the tag such as a pluralized spelling of the tag or common abbreviation. For example, the tag “New York” may have an alias of “ny”.

   Description
   : A description or definition of the tag.

3. Once you feel ready to save your changes, scroll back to the top of the page and click “Save” (the icon that looks like a floppy disk) in the upper right portion of the page.

Editing a Tag

To edit a tag, click the raw tag, tag, or alias found in the tag listing.

All other steps are identical to tag creation.

Merging Tags

Should a situation arise where two or more tags should really be one tag, you can merge tags. To merge tags, check the box next to the tags you wish to merge and hit the “merge” button in the toolbar located at the top right of the page.

You will then be presented with a list of the tags you want to merge (and how many items are using each tag) and the options to either:

- Merge the tags under another existing tag (ex: everything tagged with “h2o” and “hydro” can be merged to just “hydro”) or
- Merge the tags into a new tag (ex: everything tagged with “h2o” and “hydro” can be merged to “water”)

Click “Save” (the icon that looks like a floppy disk) in the upper right portion of the page to perform the final merge. **Note:** This cannot be undone!

Deleting Tags

To delete one or more tags, check the box next to the tag(s) you wish to delete and hit the “delete” button in the toolbar located at the top right of the page.

Contribtool

Changing the Developer URL

When accessing a tool's development page, there is a link available under the "What's next?" section on the right-hand side of the page (see the sample image below). This link should direct the user to the yourhub.org/tools page corresponding to that particular hub. But if that is not the case, the link can be changed quickly from the backend administrator interface; simply follow the instructions below.

Begin by...

1. Logging into the *Joomla!* administrative interface corresponding to your site
2. Then, navigate to "Components -> Contribtool" from the main menu bar on the top of the screen
3. Next, select parameters from the top-right corner of the page

4. Locate the "Developer URL" field and ensure that it reads https://yourhub.org/tools

5. Select "Save" from the top-right corner

XPolls

Adding a Poll

To add a new poll to you HUB, all you have to do is...

1. Log on to the *Joomla!* administrator interface (https://yourhub.org/administrator)
2. Navigate to "Components -> XPolls" on the main menu bar
3. Click the "New" button in the top right corner of your screen to add a new poll

4. Fill in the details of your poll in the fields provided

5. Select "Save"

Note: Users can only vote on the most recent open poll, but they can view results for all of the published polls.

Web Developers

Written in a book format, it contains the information a developer needs to not only understand and use HUBzero components but build extensions for a HUBzero installation. Developers will learn how to use common objects, available code libraries and utilities, and distinguish between and develop the following kinds of extensions:

- Components
- Modules
- Plugins
- Templates

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:

- (X)HTML
- Cascading Stylesheets (CSS)
- JavaScript (familiarity with the MooTools 1.11 framework is a plus)
- XML
- Model-View-Controller (MVC) design pattern
- Object-Oriented Programming

Installation

Directories & File Structure

The initial directory structure of a HUBzero install.

```
/hubzero
  /administrator
  /cache
  /components
  /images
  /includes
  /language
  /libraries
  /logs
  /media
  /modules
  /plugins
  /site
  /templates
  /tmp
  /xmlrpc
  configuration.php
  index.php
  index2.php
  htaccess.txt
  robots.txt
```

While this looks very much like a typical Joomla! 1.5 install, there are some noticeable exceptions. Some directories vital to HUBzero functionality have been added. A quick explanation of the additional directories:

/site
: This is where HUB specific data such as member pictures, files used in wiki pages, etc. is stored.

Accessing Files

Accessing via SSH

The following tutorial should help you in using SSH to connect to and from your HUBzero server(s). You should be relatively comfortable with using a terminal (also referred to as a "command-line tool") to navigate directories and manipulate files.

Warning: Most accounts do **not** have SSH/sFTP access initially. Your system administrator must grant your account access before you will be able to connect.

From a terminal type ssh <user>@<host>. You will then be prompted for a password. Both the username and password will typically be the same as the account you registered on <host>.

```
yourmachine:~ you$ ssh username@host
yourmachine:~ you$ username@host password:

host ~
```

Windows Clients

- PuTTY (a Telnet and SSH client)

Mac OSX

All versions of Mac OSX come with Terminal.app which may be found in the /Utilities directory of your /Applications directory.

Accessing via sFTP

sFTP, or secure FTP, is a program that uses SSH to transfer files. Unlike standard FTP, it encrypts both commands and data, preventing passwords and sensitive information from being transmitted in the clear over the network. It is functionally similar to FTP, but because it uses a different protocol, you can't use a standard FTP client to talk to an sFTP server, nor can you connect to an FTP server with a client that supports only sFTP.

The following tutorial should help you in using sFTP to connect to and from your HUBzero server(s).

Warning: Most accounts do **not** have SSH/sFTP access initially. Your system administrator

must grant your account access before you will be able to connect.

Graphical Clients

Using graphical SFTP clients simplifies file transfers by allowing you to transmit files simply by dragging and dropping icons between windows. When you open the program, you will have to enter the name of the host (e.g., yourhub.org) and your HUB username and password.

Windows Clients

- WinSCP
- BitKinex
- FileZilla
- PuTTY

Mac OSX Clients

- Transmit
- Fetch
- Cyberduck
- Flow
- Fugu

Command-line

You can use command line SFTP from your Unix account, or from your Mac OS X or Unix workstation. To start an SFTP session, at the command prompt, enter:

```
yourmachine:~ you$ sftp username@host
yourmachine:~ you$ username@host password:
```

host ~

Some standard commands for command-line sFTP

Command	Description
cd	Char
chmod	Char comp
chown	Char comp
dir (or ls)	List t remo
exit (or quit)	Close and e
get	Copy local

Command	Description
help (or ?)	Get h
lcd	Char
lls	See a the lo
lmkdir	Crea
ln (or symlink)	Crea comp
lpwd	Show direc
lumask	Char
mkdir	Crea
put	Copy remo
pwd	Show direc
rename	Rena
rm	Delet
rmdir	Rem direc
version	Displ
!	In Ur enter SFTP !pwd) dropp

Finding Files

Once connected to a server, by either sFTP or directly with SSH, you will need to find the web root which contains the HUB install. The web root for the production version of a HUB can be found at /www/yourhub. Typically, HUBs will also have a development version of a HUB, which can be found at /www/dev.

Once in the desired directory, file layout and directory structure follows Joomla! 1.5 conventions unless otherwise noted.

See the Installation overview for details on a typical HUBzero install's directory structure.

Direct Database Access

Accessing via command-line

The following tutorial should help you in using SSH to connect to and from your HUBzero server(s) and access the database. You should be relatively comfortable with using a terminal (also referred to as a "command-line tool") to navigate directories and manipulate files.

Warning: Most accounts do **not** have SSH/sFTP access initially. Your system administrator must grant your account access before you will be able to connect.

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

Libraries

Hubzero

Location:

`/plugins/xhub/xlibraries`

The Hubzero library contains code that is essential for a hub to run properly and altering or adding to the library without Hubzero approval is *strongly* discouraged.

Debugging

Joomla's Debugging Mode

To turn on Joomla!'s Debug mode:

- Login to the Joomla administration e.g. http:/YOURSITE/administrator/
- At the top under the **Site** menu click **Global Configuration**.
- Click the **System** tab.
- Under the **Debug Settings** section change **Debug System** to *Yes*.
- Click the **Save** button.

Debug mode will output a list of all queries that were executed in order to generate the page. This will also turn on a stack trace output for error and warning pages. Hubzero components will also have PHP error reporting turned on, allowing one to see any PHP errors that may be present.

Note: Turning on debugging mode for production (live) sites is strongly discouraged and it is recommended to be avoided if at all possible.

Illegal variable ... passed to script.

One encounters the following error:

Illegal variable _files or _env or _get or _post or _cookie or _server or _session or globals passed to script.

This error is generated when the key of a key-value pair is numeric in one of the following variables: _files or _env or _get or _post or _cookie or _server or _session or globals. An example of this would be $_POST[5] = 'value'. This is most often generated by having form elements with numeric values as names. For example:

```
<input type="text" name="5" />
```

As the error indicates, this is not allowed. Element names must include at least one non-numeric character. Examples:

```
<input type="text" name="n5" />
```

```
<input type="text" name="n_5" />
```

0.8.0

Common Tasks & Objects

Config

Joomla! Configuration

Accessing the global Joomla! site configuration:

```
$jconfig =& JFactory::getConfig();
```

Retrieving a value from the configuration:

```
echo $config->getValue('config.sitename');
```

HUB Configuration

Although rarer than accessing the global Joomla! site configuration, sometimes it is necessary to access HUB-specific configurations. This can be done as follows:

```
$xhub =& XFactory::getHub();
```

Retrieving a value from the configuration:

```
echo $xhub->getCfg('hubShortName');
```

Users & Profiles

Joomla User Object

Current User

Accessing the Joomla! User object for the current user can be done as follows:

```
$juser =& JFactory::getUser();
```

Other Users

To access user info for anyone not the current user (accepts user ID number or username):

```
$otheruser =& JUser::getInstance($id);
```

Any field from the user database table may then be accessed through the get('fieldname') method:

```
$id = $juser->get('id');
$name = $juser->get('name');
```

Object Member Variables and Parameters

These are the relevant member variables automatically generated on a call to getUser():

- id - The unique, numerical user id. Use this when referencing the user record in other database tables.
- name - The name of the user. (e.g. Vint Cerf)
- username - The login/screen name of the user. (e.g. shmuffin1979)
- email - The email address of the user. (e.g. crashoverride@hackers.com)
- password - The encrypted version of the user's password
- password_clear - Set to the user's password only when it is being changed. Otherwise, remains blank.
- usertype - The role of the user within Joomla!. (Super Administrator, Editor, etc...)
- gid - Set to the user's group id, which corresponds to the usertype.
- block - Set to '1' when the user is set to 'blocked' in Joomla!.
- registerDate - Set to the date when the user was first registered.

- lastvisitDate - Set to the date the user last visited the site.
- guest - If the user is not logged in, this variable will be set to '1'. The other variables will be unset or default values.

In addition to the member variables (which are stored in the database in columns), there are parameters for the user that hold preferences. To get one of these parameters, call the getParam() member function of the user object, passing in the name of the parameter you want along with a default value in case it is blank.

```
$user =& JFactory::getUser();
$language = $user->getParam('language', 'the default');

echo "<p>Your language is set to {$language}.</p>";
```

HUBzero Extended Profile

HUBzero comes with extended user profiles that allow for considerably more information than the standard Joomla! User. Extended fields include information about disability, gender, race, bios, picture, etc. To access an extended profile, use the XProfile object and load() method (accepts user ID number or username).

```
// Import the needed library
ximport('Hubzero_User_Profile');

// Instantiate a new profile object
$profile = new Hubzero_User_Profile();

// Load the profile
$profile->load( $id );
```

Any field from the user database table may then be accessed through the get('fieldname') method:

```
$email = $profile->get('email');
$name = $profile->get('name');
```

Multi-option fields such as disability will return arrays.

Checking if a User is logged in

Checking if a user is currently logged in can be done as follows:

```
// Call the user object
$juser =& JFactory::getUser();

// If 'guest' is true, they are logged OUT
// If 'guest' is false, they are logged IN
if ($juser->get('guest')) {
 return false;
}
```

Privileges

Not all authenticated users are given equal rights. For instance, a Super Administrator may be able to edit anyone's content, while a Publisher may only be able to edit their own. The authorize() member function can be used to determine if the current user has permission to do a certain task. The first parameter is used to identify which component or function we wish to authenticate against. The second represents the task. The third and fourth are optional; they further break the permissions down into record types and ownership respectively.

In Joomla! 1.5, the rights for all of the core components are stored in libraries/joomla/user/authorization.php. These are available to all extensions wherever authentication is required. If the permission scheme of the Content component suits your extension's needs, you can use code similar to the following to determine what functions to give to a specific user.

```
$user =& JFactory::getUser();

if ($user->authorize('com_content', 'edit', 'content', 'all')) {
 echo "<p>You may edit all content.</p>";
} else {
 echo "<p>You may not edit all content.</p>";
}

if ($user->authorize('com_content', 'publish', 'content', 'own')) {
 echo "<p>You may publish your own content.</p>";
} else {
 echo "<p>You may not publish your own content.</p>";
}
```

The permissions for core functions may not be suitable for your extension. If this is the case, you can create your own permissions. You will probably want to add this code in a place where it will always be executed, such as the beginning of the component you are building or in a systemwide plugin. First, you need to get an authorization object using the getACL() member function of JFactory. This works like getUser() in that it only creates one authorization object during any particular Joomla! request. Once you have this object, call the addACL() member function to add permissions. Pass in the name of your component or function, the task name, the string 'users', and the user type (in lowercase) respectively. If you want to also define record sets and ownership, pass those in as an additional two parameters.

Note that in Joomla! 1.5, permissions are not inherited. For example, if you give an Administrator the right to edit content, Super Administrators do not automatically get this right; you must grant it separately.

```
$auth =& JFactory::getACL();

$auth->addACL('com_userinfo15', 'persuade', 'users', 'super administra
tor');
$auth->addACL('com_userinfo15', 'persuade', 'users', 'administrator');
$auth->addACL('com_userinfo15', 'persuade', 'users', 'manager');

$user =& JFactory::getUser();

if ($user->authorize('com_userinfo15', 'persuade')) {
 echo "<p>You may persuade the system to do what you wish.</p>";
} else {
 echo "<p>You are not very persuasive.</p>";
}
```

Group Memberships

Sometimes you may have a component or plugin that is meant to be accessed by members of a certain group or displays specific data based on membership in certain groups.

```
// Get the user object
$juser =& JFactory::getUser();

// Include a needed HUBzero library
ximport('Hubzero_User_Helper');

// Get the groups of the current logged-in user
$user_groups = Hubzero_User_Helper::getGroups( $juser->get('id') );
```

The getGroups() method is passed a user ID and returns an array of objects if any group memberships are found. It will return false if no group memberships are found. Each object contains data specifying the user's status within the group, among other things.

```
Array (
    [0] => stdClass Object (
  [published] => 1
  [cn] => greatgroup
  [description] => A Great Group
  [registered] => 1
  [regconfirmed] => 1
  [manager] => 0
 )
    [1] => stdClass Object (
  [published] => 1
  [cn] => mygroup
  [description] => My Group
  [registered] => 1
  [regconfirmed] => 1
  [manager] => 1
 )
)
```

- published - 0 or 1, the published state of the group
- cn - string, the group alias
- description - string, the group title
- registered - 0 or 1, if the user applied for membership to this group (only 0 if the user was invited)
- regconfirmed - 0 or 1, if the user's membership application has been accepted (automatically 1 for invitees)
- manager - 0 or 1, if the user is a manager of this group

Database

Overview

Joomla! has been built with the ability to use several different kinds of SQL-database-systems and to run in a variety of environments with different table-prefixes. In addition to these functions, the class automatically creates the database connection. Besides instantiating the object, at a minimum, you only need 2 lines of code to get a result from the database in a variety of formats. Using the Joomla! database layer ensures a maximum of compatibility and flexibility for your extension.

This tutorial looks at how to set and execute various queries.

Preparing The Query

```
// Get a database object
$db =& JFactory::getDBO();

$query = "SELECT * FROM #__example_table WHERE id = 999999;";
$db->setQuery($query);
```

First we instantiate the database object, then we prepare the query. You can use the normal SQL-syntax, the only thing you have to change is the table-prefix. To make this as flexible as possible, Joomla! uses a placeholder for the prefix, the "#__". In the next step, the $db->setQuery(), this string is replaced with the correct prefix.

Now, if we don't want to get information from the database, but insert a row into it, we need one more function. Every string-value in the SQL-syntax should be quoted. For example, MySQL uses back-ticks `` for names and single quotes '' for values. Joomla! has some functions to do this for us and to ensure code compatibility between different databases. We can pass the names to the function $db->nameQuote($name) and the values to the function $db->Quote($value).

A fully quoted query example is:

```
$query = "
  SELECT *
    FROM ".$db->nameQuote('#__example_table')."
    WHERE ".$db->nameQuote('id')." = ".$db->quote('999999').";
  ";
```

Whatever we want to do, we have to set the query with the $db->setQuery() function. Although you could write the query directly as a parameter for $db->setQuery(), it's commonly done by first saving it in a variable, normally $query, and then handing this variable over. This helps writing clean, readable code.

setQuery($query)

The setQuery($query) method sets up a database query for later execution either by the query() method or one of the Load result methods.

```
$db =& JFactory::getDBO();
$query = "/* some valid sql string */";
$db->setQuery($query);
```

Note: The parameter $query must be a valid SQL string, it can either be added as a string parameter or as a variable; generally a variable is preferred as it leads to more legible code and can help in debugging.

setQuery() also takes three other parameters: $offset, $limit - both used in list pagination; and $prefix - an alternative table prefix. All three of these variables have default values set and can usually be ignored.

Executing The Query

To execute the query, Joomla! provides several functions, which differ in their return value.

Basic Query Execution

The query() method is the the basic tool for executing sql queries on a database. In Joomla! it is most often used for updating or administering the database simple because the various load methods detail on this page have the query step built in to them.

The syntax is very straightforward:

```
$db =& JFactory::getDBO();
$query = "/* some valid sql string */";
$db->setQuery($query);
$result = $db->query();
```

Note: $db->query() returns an appropriate database resource if successful, or FALSE if not.

Query Execution Information

- getAffectedRows()
- explain()
- insertid()

Insert Query Execution

- insertObject()

Query Results

The database class contains many methods for working with a query's result set.

Single Value Result

loadResult()

Use loadResult() when you expect just a single value back from your database query.

id	name	email	username
1	John Smith	johnsmith@example.com	johnsmith
2	Magda Hellman	magda_h@example.com	magdah
3	Yvonne de Gaulle	ydg@example.com	ydegaulle

This is often the result of a 'count' query to get a number of records:

```
$db =& JFactory::getDBO();
$query = "
  SELECT COUNT(*)
    FROM ".$db->nameQuote('#__my_table')."
    WHERE ".$db->nameQuote('name')." = ".$db->quote($value).";
  ";
$db->setQuery($query);
$count = $db->loadResult();
```

or where you are just looking for a single field from a single row of the table (or possibly

a single field from the first row returned).

```
$db =& JFactory::getDBO();
$query = "
  SELECT ".$db->nameQuote('field_name')."
    FROM ".$db->nameQuote('#__my_table')."
    WHERE ".$db->nameQuote('some_name')." = ".$db->quote($some_va
lue).";
  ";
$db->setQuery($query);
$result = $db->loadResult();
```

Single Row Results

Each of these results functions will return a single record from the database even though there may be several records that meet the criteria that you have set. To get more records you need to call the function again.

id	name	email	username
1	John Smith	johnsmith@example.com	johnsmith
2	Magda Hellman	magda_h@example.com	magdah
3	Yvonne de Gaulle	ydg@example.com	ydegaulle

loadRow()

loadRow() returns an indexed array from a single record in the table:

```
...
$db->setQuery($query);
$row = $db->loadRow();
print_r($row);
```

will give:

```
Array (
 [0] => 1
 [1] => John Smith
 [2] => johnsmith@example.com
```

```
 [3] => johnsmith
)
```

You can access the individual values by using:

```
$row['index'] // e.g. $row['2']
```

Note:

1. The array indices are numeric starting from zero.
2. Whilst you can repeat the call to get further rows, one of the functions that returns multiple rows might be more useful

loadAssoc()

loadAssoc() returns an associated array from a single record in the table:

```
$db->setQuery($query);
$row = $db->loadAssoc();
print_r($row);
```

will give:

```
Array (
 [id] => 1
 [name] => John Smith
 [email] => johnsmith@example.com
 [username] => johnsmith
)
```

You can access the individual values by using:

```
$row['name'] // e.g. $row['name']
```

Whilst you can repeat the call to get further rows, one of the functions that returns multiple rows might be more useful

loadObject()

loadObject() returns a PHP object from a single record in the table:

```
$db->setQuery($query);
$result = $db->loadObject();
print_r($result);
```

will give:

```
stdClass Object (
 [id] => 1
 [name] => John Smith
 [email] => johnsmith@example.com
 [username] => johnsmith
)
```

You can access the individual values by using:

```
$row->index // e.g. $row->email
```

Whilst you can repeat the call to get further rows, one of the functions that returns multiple rows might be more useful

Single Column Results

Each of these results functions will return a single column from the database.

id	name	email	username
1	John Smith	johnsmith@example.com	johnsmith
2	Magda Hellman	magda_h@example.com	magdah
3	Yvonne de Gaulle	ydg@example.com	ydegaulle

loadResultArray()

loadResultArray() returns an indexed array from a single column in the table:

```
$query = "
  SELECT name, email, username
    FROM . . . ";

$db->setQuery($query);
$column= $db->loadResultArray();
print_r($column);
```

will give:

```
Array (
 [0] => John Smith
 [1] => Magda Hellman
 [2] => Yvonne de Gaulle
)
```

You can access the individual values by using:

```
$column['index'] // e.g. $column['2']
```

Note:

1. The array indices are numeric starting from zero.
2. loadResultArray() is equivalent to loadResultArray(0)

loadResultArray($index)

loadResultArray($index) returns an indexed array from a single column in the table:

```
$query = "
  SELECT name, email, username
    FROM . . . ";

$db->setQuery($query);
$column= $db->loadResultArray(1);
print_r($column);
```

will give:

```
Array (
 [0] => johnsmith@example.com
 [1] => magda_h@example.com
 [2] => ydg@example.com
)
```

You can access the individual values by using:

```
$column['index'] // e.g. $column['2']
```

loadResultArray($index) allows you to iterate through a series of columns in the results

```
$db->setQuery($query);
for ( $i = 0; $i loadResultArray($i);
  print_r($column);
}
```

will give:

```
Array ( [0] => John Smith [1] => Magda Hellman [2] => Yvonne de G
aulle )
Array ( [0] => johnsmith@example.com [1] => magda_h@example.com [
2] => ydg@example.com )
Array ( [0] => johnsmith [1] => magdah [2] => ydegaulle )
```

The array indices are numeric starting from zero.

Multi-Row Results

Each of these results functions will return multiple records from the database.

id	name	email	username
1	John Smith	johnsmith@example.com	johnsmith
2	Magda Hellman	magda_h@example.com	magdah
3	Yvonne de Gaulle	ydg@example.com	ydegaulle

loadRowList()

loadRowList() returns an indexed array of indexed arrays from the table records returned by the query:

```
$db->setQuery($query);
$row = $db->loadRowList();
print_r($row);
```

will give:

```
Array (
 [0] => Array ( [0] => 1 [1] => John Smith [2] => johnsmith@examp
le.com [3] => johnsmith )
 [1] => Array ( [0] => 2 [1] => Magda Hellman [2] => magda_h@exam
ple.com [3] => magdah )
 [2] => Array ( [0] => 3 [1] => Yvonne de Gaulle [2] => ydg@examp
le.com [3] => ydegaulle )
)
```

You can access the individual values by using:

```
$row['index'] // e.g. $row['2']
```

and you can access the individual values by using:

```
$row['index']['index'] // e.g. $row['2']['3']
```

The array indices are numeric starting from zero.

loadAssocList()

loadAssocList() returns an indexed array of associated arrays from the table records returned by the query:

```
$db->setQuery($query);
$row = $db->loadAssocList();
print_r($row);
```

will give:

```
Array (
 [0] => Array ( [id] => 1 [name] => John Smith [email] => johnsmi
th@example.com [username] => johnsmith )
 [1] => Array ( [id] => 2 [name] => Magda Hellman [email] => magd
a_h@example.com [username] => magdah )
 [2] => Array ( [id] => 3 [name] => Yvonne de Gaulle [email] => y
dg@example.com [username] => ydegaulle )
)
```

You can access the individual rows by using:

```
$row['index'] // e.g. $row['2']
```

and you can access the individual values by using:

```
$row['index']['column_name'] // e.g. $row['2']['email']
```

loadAssocList($key)

loadAssocList($key) returns an associated array - indexed on 'key' - of associated arrays from the table records returned by the query:

```
$db->setQuery($query);
$row = $db->loadAssocList('username');
print_r($row);
```

will give:

```
Array (
 [johnsmith] => Array ( [id] => 1 [name] => John Smith [email] =>
 johnsmith@example.com [username] => johnsmith )
 [magdah] => Array ( [id] => 2 [name] => Magda Hellman [email] =>
 magda_h@example.com [username] => magdah )
 [ydegaulle] => Array ( [id] => 3 [name] => Yvonne de Gaulle [ema
il] => ydg@example.com [username] => ydegaulle )
)
```

You can access the individual rows by using:

```
$row['key_value'] // e.g. $row['johnsmith']
```

and you can access the individual values by using:

```
$row['key_value']['column_name'] // e.g. $row['johnsmith']['email
']
```

Note: Key must be a valid column name from the table; it does not have to be an Index or a Primary Key. But if it does not have a unique value you may not be able to retrieve results reliably.

loadObjectList()

loadObjectList() returns an indexed array of PHP objects from the table records returned by the query:

```
$db->setQuery($query);
$result = $db->loadObjectList();
print_r($result);
```

will give:

```
Array (
 [0] => stdClass Object ( [id] => 1 [name] => John Smith
     [email] => johnsmith@example.com [username] => johnsmith )
 [1] => stdClass Object ( [id] => 2 [name] => Magda Hellman
     [email] => magda_h@example.com [username] => magdah )
 [2] => stdClass Object ( [id] => 3 [name] => Yvonne de Gaulle
     [email] => ydg@example.com [username] => ydegaulle )
)
```

You can access the individual rows by using:

```
$row['index'] // e.g. $row['2']
```

and you can access the individual values by using:

```
$row['index']->name // e.g. $row['2']->email
```

loadObjectList('key')

loadObjectList('key') returns an associated array - indexed on 'key' - of objects from the table records returned by the query:

```
$db->setQuery($query);
$row = $db->loadObjectList('username');
print_r($row);
```

will give:

```
Array (
 [johnsmith] => stdClass Object ( [id] => 1 [name] => John Smith
     [email] => johnsmith@example.com [username] => johnsmith )
 [magdah] => stdClass Object ( [id] => 2 [name] => Magda Hellman
     [email] => magda_h@example.com [username] => magdah )
 [ydegaulle] => stdClass Object ( [id] => 3 [name] => Yvonne de G
aulle
     [email] => ydg@example.com [username] => ydegaulle )
)
```

You can access the individual rows by using:

```
$row['key_value'] // e.g. $row['johnsmith']
```

and you can access the individual values by using:

```
$row['key_value']->column_name // e.g. $row['johnsmith']->email
```

Note: Key must be a valid column name from the table; it does not have to be an Index or a Primary Key. But if it does not have a unique value you may not be able to retrieve results reliably.

Misc Result Set Methods

getNumRows()

getNumRows() will return the number of result rows found by the last query and waiting to be read. To get a result from getNumRows() you have to run it after the query and before you have retrieved any results.

```
$db->setQuery($query);
$db->query();
$num_rows = $db->getNumRows();
print_r($num_rows);
$result = $db->loadRowList();
```

will give:

3

Note: if you run getNumRows() after loadRowList() - or any other retrieval method - you may get a PHP Warning.

JTable

Overview

The JTable class is an implementation of the Active Record design pattern. It is used throughout Joomla! for creating, reading, updating, and deleting records in the database table.

When properly extended, JTable gives you all of the basic functions you need for managing and retrieving records in a database table. Member functions take care of the rest when you add member variables, the table name, and the key column.

Writing an extension of JTable

To use JTable, create an extension of the class. In this example, we have a database table containing recipes.

```
<?php

defined('_JEXEC') or die();

class TableRecipes extends JTable
{
 var $id = null;
 var $ingredients = null;
 var $instructions = null;
 var $serves = null;
 var $difficulty = null;
 var $prep_time = null;
 var $cook_time = null;
 var $published = 0;

 function __construct(&$db)
 {
  parent::__construct( '#__recipes', 'id', $db );
 }
}
```

When naming your class extension, the convention is to prefix it with 'Table', then follow with a CamelCased version of the table's name. All of the member variables of your class should match the column names in the database. The default values should be valid according to the table schema. For instance, if you have columns that are NOT NULL, you must use a value

other than 'null' as the default.

Finally, create a constructor for the class that accepts a reference to the current database instance. This will call the parent constructor which needs the name of the table, the name of the primary key column, and the database instance. The name of the table uses #__ instead of jos_, as the administrator can pick any table prefix desired during Joomla! installation.

If you were using this class as a part of a component called 'Recipes', you would place this code in the file /administrator/components/com_recipes/tables/recipes.php.

Using a JTable class extension

Once the table class is in place, you can use it in any Joomla! extension. To include the file, place this line in your extension's source code (use com_nameofyourcomponent in place of com_recipes):

```
JTable::addIncludePath(JPATH_ADMINISTRATOR.DS.'components'.DS.'com_recipes'.DS.'tables');
```

To get an instance of the object, use this code:

```
$row =& JTable::getInstance('recipes', 'Table');
```

Notice that the lowercase version of the suffix of your class name is used as the first parameter, with the prefix 'Table' as the second. Also, the getInstance() member function of JTable returns the object by reference instead of value.

In a model class (extends JModel) you can also use:

```
$row =& $this->getTable('recipes');
```

Notice that if you have not used the standard naming convention, you can supply the class prefix as the optional second parameter.

Create/Update

In a typical situation, you will have an HTML form submitted by the user which PHP will interpret for you as an associative array. The JRequest class in Joomla! has functions ready to assist with retrieving this data safely. Use JRequest::get('post') to retrieve all of the elements in the HTTP POST request as a sanitized array.

Once you have this array, you can pass it into the bind() method of JTable. Doing this will match the associated items of the array with member variables of the class. In the following example, the array is retrieved from JRequest::get('post') and immediately passed into bind().

```
if (!$row->bind( JRequest::get( 'post' ) )) {
 return JError::raiseWarning( 500, $row->getError() );
}
```

If bind() fails, you want to stop the application and explain the failure before your extension attempts to send the data. The raiseWarning() function of JError allows you to stop Joomla!, while the getError() function returns the error message stored in the JTable object.

When binding succeeds and your object is ready, call the store() function. Again, if something goes wrong, stop the application and explain why.

```
if (!$row->store()) {
 JError::raiseError(500, $row->getError() );
}
```

Note:

- If any member variables of your JTable object are null when store() is called, they are ignored by default. This allows you to update specific columns of your table, while leaving the others untouched. If you wish to override this behavior to ensure that all columns have a value, pass true into store().
- The JTable::bind() and JRequest::get() functions do not enforce data types. If you need a column to be a specific type (for instance, integer), you need to add this logic to your code before calling store().

Read

To load a specific row of the database with JTable, pass the key into the load() member function.

```
$row->load( $id );
```

This relies on the key column you specified in the second parameter of parent::__construct() when you extended JTable.

Delete

Like read(), delete() allows you to destroy a specific row in the table based on the key specified earlier.

```
$row->delete( $id );
```

If you want to delete multiple rows at once, you will need to write the query manually.

Tags

Overview

The Tag class is a set of tools for adding, removing, editing, and displaying tags on objects. It is used throughout HUB installations for adding tags to such things as resources, users, events, and more.

When properly extended, Tags gives you all of the basic functions you need for managing and retrieving tag records in the database table.

All tags are stored within a single table called "#__tags". The information that associates a particular tag to a specific user, event or group, is stored in a table called "#__tags_object". Storing the association data separate from the tag itself allows for a tag to be represented once but be connected to multiple items. If that tag is ever changed for any reason, it will be represented the same regardless of what object it is attached to.

The #__tags_object table stores, among other things, such data as the unique ID of the tag, the unique ID of the object being tagged, and what component (or, potentially, table) that object belongs to.

id	objectid	tagid	tbl
1	77	6	resources
2	77	6	events

Here we have two entries that both link to a tag with an ID of "6" and both with object IDs of "77". One entry is a resource and the other is an event. The "tbl" field is the most important distinguishing factor; This allows us to have multiple objects with the same object ID, linking to the same tag but not create a conflict.

Writing an extension of Tags

To use Tags, create an extension of the class. In this example, we're adding tags to our "com_example" objects.

```
<?php
// Check to ensure this file is included in Joomla!
defined('_JEXEC') or die( 'Restricted access' );

require_once( JPATH_ROOT.DS.'components'.DS.'com_tags'.DS.'helpers'.DS
.'handler.php' );

class ExampleTags extends TagsHandler
{
```

```
 public function __construct( $db )
 {
  // The database connection object
  $this->_db  = $db;
  // A unique name
  $this->_tbl = 'example';
 }
}
```

When naming your class extension, the convention is to have a CamelCased version of the component's name suffixed with "Tags".

Finally, create a constructor for the class that accepts a reference to the current database instance and the name to be used to uniquely identify tag data as belonging to your specific component.

Using a Tag class extension

Once the class is created and in place, it can be included and instantiated

Create/Update

```
// Retrieve posted tags (comma delimitated string)
$tags = trim(JRequest::getVar( 'tags', '' ));

// Get the database object
$database =& JFactory::getDBO();

// Instantiate the tagging class
$et = new ExamplesTags( $database );

// Tag the object
$et->tag_object( $juser->get('id'), $object_id, $tags );
```

This method is the same for both adding tags to a previously untagged object and updating the existing list of tags on an object.

Read

get_tag_cloud($showsizes, $showadmintags, $object_id)

Returns a string of comma-separated tags.

```
// Get the database object
$database =& JFactory::getDBO();

// Instantiate the tagging class
$et = new ExamplesTags( $database );

// Get a tag cloud (HTML List)
$tags = $et->get_tag_cloud( $showsizes, $showadmintags, $object_i
d );
print_r($tags);
```

will give:

```
My Tag, Your Tag, Their Tag
```

get_tag_cloud($showsizes, $showadmintags, $object_id)

Returns a tag cloud, derived of a an HTML list. Each tag is linked to the Tags component and comprises one list item. A CSS class of "tags" on the list allows for styling.

```
// Get the database object
$database =& JFactory::getDBO();

// Instantiate the tagging class
$et = new ExamplesTags( $database );

// Get a string of tags separated by commas
$tags = $et->get_tag_string( $object_id );
print_r($tags);
```

will give:

```
<ol class="tags">
 <li><a href="/tags/mytag">My Tag</a></li>
 <li><a href="/tags/yourtag">Your Tag</a></li>
 <li><a href="/tags/theirtag">Their Tag</a></li>
</ol>
```

get_tags_on_object($object_id)

Returns an array of associative arrays.

```
// Get the database object
$database =& JFactory::getDBO();

// Instantiate the tagging class
$et = new ExamplesTags( $database );

// Get a string of tags separated by commas
$tags = $et->get_tags_on_object( $object_id );
print_r($tags);
```

will give:

```
Array (
 [0] => Array (
   [tag] => 'mytag'
   [raw_tag] => 'My Tag'
   [tagger_id] => 32
   [admin] => 0
 )
 [1] => Array (
   [tag] => 'yourtag'
   [raw_tag] => 'Your Tag'
   [tagger_id] => 32
   [admin] => 0
 )
 [2] => Array (
   [tag] => 'theirtag'
   [raw_tag] => 'Their Tag'
   [tagger_id] => 32
   [admin] => 0
  )
```

```
    )
```

Using the Tag Editor plugin

To make adding tags and editing a list of existing tags in a form, HUBzero offers a Tag Editor plugin. To use the plugin in a view, do the following:

```
// Load the plugin
JPluginHelper::importPlugin( 'hubzero' );
$dispatcher =& JDispatcher::getInstance();

// Trigger the event
$tf = $dispatcher->trigger( 'onGetMultiEntry', array(array('tags','tag
s','actags','',$tags)) );

// Output
if (count($tf) > 0) {
 echo $tf[0];
} else {
 echo '<input type="text" name="tags" value="'. $tags .'" />';
}
```

The first parameter passed ('tags') tells the plugin that you wish to display a tags autocompleter. The nest parameter is the name of the input field. The third is the ID of the input field. The fourth is any CSS class you wish to assign to the input. The $tags variable here must be a string of comma-separated tags.

Search

Retrieving GET & POST data

JRequest 'getVar' method

To retrieve GET/POST request data, Joomla! uses the getVar method of the JRequest class (JRequest::getVar()).

Retrieving Data

If you have a form variable named 'address', you would want to use this code to get it:

```
$address = JRequest::getVar('address');
```

Unless other parameters are set, all HTML and trailing whitespace will be filtered out.

The DEFAULT Parameter

If you want to specify a default value in the event that 'address' is not in the request or is unset, use this code:

```
$address = JRequest::getVar('address', 'Address is empty');
echo $address;  // Address is empty
```

The SOURCE Parameter

Frequently, you will expect your variable to be found in a specific portion of the HTTP request (POST, GET, etc...). If this is the case, you should specify which portion; this will slightly increase your extension's security. If you expect 'address' to only be in POST, use this code to enforce that:

```
$address = JRequest::getVar('address', 'default value goes here', 'pos
t');
```

The VARIABLE TYPE Parameter

The fourth parameter of getVar() can be used to specify certain filters to force validation of specific value types for the variable.

```
$address = JRequest::getVar('address', 'default value goes here', 'pos
t','variable type');
```

Here is a list of types you can validate:

- INT
- INTEGER
- FLOAT
- DOUBLE
- BOOL
- BOOLEAN
- WORD
- ALNUM
- CMD
- BASE64
- STRING
- ARRAY
- PATH
- USERNAME

The FILTER MASK Parameter

Finally, there are some mask constants you can pass in as the fifth parameter that allow you to bypass portions of the filtering:

```
$address = JRequest::getVar('address', 'default value goes here', 'pos
t','validation type','mask type');
```

- JREQUEST_NOTRIM - prevents trimming of whitespace
- JREQUEST_ALLOWRAW - bypasses filtering
- JREQUEST_ALLOWHTML - allows most HTML. If this is not passed in, HTML is stripped out by default.

Constants

Joomla! Constants

These constants are defined for use in Joomla and extensions:

DS	Directory separator. "/"
JPATH_ADMINISTRATOR	The path to the administrator folder.
JPATH_BASE	The path to the installed Joomla! site.
JPATH_CACHE	The path to the cache folder.
JPATH_COMPONENT	The path to the current component being executed.
JPATH_CONFIGURATION	The path to folder containing the configuration.php file.
JPATH_INSTALLATION	The path to the installation folder.
JPATH_LIBRARIES	The path to the libraries folder.
JPATH_PLUGINS	The path to the plugins folder.
JPATH_ROOT	The path to the installed Joomla! site.
JPATH_SITE	The path to the installed Joomla! site.
JPATH_THEMES	The path to the templates folder.
JPATH_XMLRPC	The path to the XML-RPC Web service folder.

Note: These paths are the absolute paths of these locations within the file system, NOT the path you'd use in a URL.

For URL paths, try using JURI::base.

Extensions (general)

Overview

Joomla! already is a rich featured content management system but if you're building a website with Joomla! and you need extra features which aren't available in Joomla! by default, you can easily extend it with extensions. There are five types of extensions for Joomla!: 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.

Hubzero Components

- com_answers
- com_blog
- com_citations
- com_contribtool
- com_contribute
- com_documentation
- com_events
- com_features
- com_feedback
- com_groups
- com_hub
- com_infrastructure
- com_jobs
- com_kb
- com_meetings
- com_members
- com_myhub
- com_register
- com_resources
- com_sef
- com_store

- com_support
- com_tags
- com_tools
- com_topics (alternate name for com_wiki)
- com_usage
- com_whatsnew
- com_wiki (alternate name for com_topics)
- com_wishlist
- com_xflash
- com_ximport
- com_xpoll (supplants Joomla's com_poll)
- com_xsearch (supplants Joomla's com_search)

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.

Hubzero Modules (front-end)

- mod_events_cal
- mod_events_latest
- mod_featuredmember
- mod_featuredquestion
- mod_featuredresource
- mod_feed_youtube
- mod_findresources
- mod_mycontributions
- mod_myfavorites
- mod_mygroups
- mod_mymeetings

- mod_mymessages
- mod_mypoints
- mod_myprofile
- mod_myquestions
- mod_mysessions
- mod_mysubmissions
- mod_mytickets
- mod_mytools
- mod_mywishes
- mod_notices
- mod_polltitle
- mod_popularfaq
- mod_popularquestions
- mod_quicktips
- mod_quotes
- mod_randomquote
- mod_rapid_contact
- mod_recentquestions
- mod_reportproblems
- mod_resourcemenu
- mod_slideshow
- mod_sliding_panes
- mod_spotlight
- mod_toptags
- mod_twitterfeed
- mod_whatsnew
- mod_wishvoters
- mod_xflash
- mod_xlogin
- mod_xlogin_mini
- mod_xpoll
- mod_xsearch
- mod_xwhosonline

Hubzero Modules (back-end/administrative)

- mod_dashboard

Plugins

Joomla! 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. Joomla! plugins enable you to execute code in response to certain events, either Joomla! core events or custom events that are triggered from your own code. This is a powerful way of extending the basic Joomla! functionality.

Hubzero Plugins

- content
    - xhubtags
- groups
    - forum
    - members
    - messages
    - resources
    - wiki
    - wishlist
- members
    - blog
    - contributions
    - favorites
    - groups
    - messages
    - points
    - resources
    - resume
    - topics
    - usage
- resources
    - citations
    - favorites
    - questions
    - recommendations
    - related
    - reviews
    - share
    - supportingdocs
    - usage
    - versions
    - wishlist
- support
    - xfeed
    - xhub
- system
    - xfeed
    - xhub
- tageditor
    - autocompleter
- tags
    - answers
    - blogs
    - events

        - groups
        - members
        - resources
        - support
        - topics
- usage
        - chart
        - domainclass
        - domains
        - maps
        - overview
        - partners
        - region
        - tools
- whatsnew
        - content
        - events
        - kb
        - resources
        - topics
- xauthentication
        - hzldap
- xhub
        - xlibrary
- xmessage
        - email
        - handler
        - im
        - internal
        - rss
        - smstxt
- xsearch
        - answers
        - blogs
        - content
        - events
        - kb
        - members
        - resources
        - tags
        - topics

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.

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 Joomla! 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 Joomla! installation and check the components menu, the module manager and the plugin manager. Joomla! comes with a number of core components, modules and plugins. By checking what theyâ€™re doing, the difference between the three types of building blocks should become clear. You can also check out the official Joomla! extensions page. Browse through the extension categories and youâ€™ll be amazed about the extension possibilities you have for your site.

Installing

Installing From Package

Warning: Unlike a typical Joomla! 1.5 install, 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.

Joomla! 1.5 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 Joomla!. 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/{ExtensionType}/ directory tree. Note: Any language files packaged with the module will be moved to their respective sub-directories of the /yourhub/language/ directory.

Installing From Directory

Joomla! 1.5 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/administrator/components/com_installer/module/yourmodulename directory of Joomla!.

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/administrator/components/com_installer/module/yourmodulename.
6. Click the "Install" button to complete the installation. The appropriate module files will be moved to the /yourhub/modules/ directory tree. Note: Any language files packaged with the module will be moved to their respective sub-directories of the /yourhub/language/ directory.

Installing By Hand

Installing an extension by hand requires a few more steps than the Joomla! 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/{ExtensionType}/ directory. Any language files associated with the extension must be copied to their respective sub-directories of the /yourhub/language/ directory.

   Components are unique in that they will typically have files installed in two locations: /components and /administrator/components.

Extension Type	Install Location
Component	/yourhub/components/{ExtensionName} /yourhub/administrator/components/{ExtensionName}
Module	/yourhub/modules/{ExtensionName}
Plugin	/yourhub/plugins/{PluginType}/
Template	/yourhub/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 #__components(
 `id`,
 `name`,
 `link`,
 `menuid`,
 `parent`,
 `admin_menu_link`,
 `admin_menu_alt`,
 `option`,
 `ordering`,
 `admin_menu_img`,
 `iscore`,
 `params`,
 `enabled`
)
VALUES(
 '',
 'My Component',
 '',
 0,
 0,
 'option=com_mycomponent',
 'My Component',
 'com_mycomponent',
 0,
 'js/ThemeOffice/component.png',
 0,
 '',
 1
);
```

   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/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 #__plugins(
 `id`,
 `name`,
 `element`,
 `folder`,
 `access`,
 `ordering`,
 `published`,
 `iscore`,
 `client_id`,
 `checked_out`,
 `checked_out_time`,
 `params`
)
VALUES(
 '',
 'System - Hello World',
 'helloworld',
 'system',
 0,
 1,
 1,
 0,
 0,
 0,
 '0000-00-00 00:00:00',
 ''
```

```
);
```

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.

Uninstalling

Overview

If you wish to uninstall an extension on your Joomla! 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 Joomla! 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.

Parameters

Overview

Coming soon.

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 JText::_('MYLINE'); ?></strong>
```

```
<!-- mycomponent layout -->
<div>
 <!-- Load the module -->
 <php echo XModuleHelper::renderModule('mymodule'); ?>
 <!-- Translate some component text -->
 <php echo JText::_('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 JText::_('MOD_MYMODULE_MYLINE'); ?></strong>
```

```
<!-- mycomponent view -->
<div>
 <!-- Load the module -->
 <php echo $this->Widgets()->renderWidget('mywidget'); ?>
 <!-- Translate some module text -->
 <php echo JText::_('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 (JText) 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 JText::_("MOD_EXAMPLE_MY_LINE"); ?></p>
```

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

Further Help

For further help with language files, including creating and distributing your own translations to existing extensions, see Joomla!'s documentation.

Components

Overview

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.

Throughout these articles, we will be using {ComponentName} to represent the name of a component that is variable, meaning the actual component name is chosen by the developer. Notice also that case is important. {componentname} will refer to the lowercase version of {ComponentName}, eg. "CamelCasedController" -> "camelcasedcontroller". Similarly, {ViewName} and {viewname}, {ModelName} and {modelname}, {ControllerName} and {controllername}.

We also strongly encourage developers to rake a look at Joomla!'s documentation.

Directory Structures & 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 #__components table will not appear in the administrative list of available components.

To illustrate the typical component directory structures and files:

```
/hubzero
  /administrator
    /components
      /com_example
```

```
...
/components
  /com_example
      /models
        foo.php
      /views
        /index
          /tmpl
            default.php
            default.xml
    controller.php
    example.php
    router.php
```

In the above example, all component related files and sub-directories are split between the administrator components and front-end components. In both cases, the files are contained within directories titled "com_example". Some directories and files are optional but, for this example, we've included the most common setup.

The file structure in the administrative portion of the component is exactly the same as in the front side. Note that the view, models, controllers etc. of the front and admin parts are completely separated, and have nothing to do with each other - the front part and the admin part can be thought of as two different components! A view in the /administrator/components/com_example folder may have a counterpart with the same name in the /components/com_example folder, yet the two views have nothing in common but their name.

Directory & File Explanation

/com_{componentname}/{componentname}.php
This is the component's main file and entry point *for the front-end part*.
/com_{componentname}/controller.php
This file holds the default frontend controller, which is a class called "{ComponentName}Controller". This class must extend the base class "JController".
/com_{componentname}/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}/view.html.php
This file is the entry point for the view {ViewName}. It should declare the class

{ComponentName}View{ViewName}. This class must extend the base class "JView".

/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}/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 "JModel". 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}/controllers

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

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

Naming Conventions

Classes

The model, view and controller files use the jimport function to load classes from the Joomla! framework, JModel, JView and JController, respectively. Each class is then extended with a new class specific to the component.

The base controller class for the site is named {ComponentName}Controller. For the administrative section, an "s" is added to the ComponentName, giving {ComponentName}sController. Classnames for additional controllers found within the controllers/ subdirectory are {ComponentName}Controller{ControllerName} for site/ and {ComponentName}sController{ControllerName} for admin/.

The view class is named {ComponentName}View{ViewName}.

The model class is named {ComponentName}Model{ModelName}. Remember that the

{ModelName} and the {ViewName} should be the same.

Reserver Words

There are reserved words, which can't be used in names of classes and components.

An example is word "view" (in any case) for view class (except "view" that must be second part of that class name). Because first part of view class name is the same as controller class name, controller class name also can't contain word "view". And because of convension (although violating of it won't produce an error) controller class name must contain component name, so component name also can't contain word "view". So components can't be named "com_reviews", or if thay are, thay must violate naming convention and have different base controller class name (or have some other hacks).

Examples

Here we have a basic front-end component that simply displays a "Hello, World!" message. We present it using both standard Joomla! 1.5 methods and HUBzero methods, which differ in a few key ways. Note, however, that despite any differences from standard Joomla! methods, all HUBzero methods will still work on a stock Joomla! 1.5 install.

Download: Hello World component (Joomla! method)

Download: Hello World component (HUBzero method)

Installation

Installing

See Installing Extensions for details.

Uninstalling

See Uninstalling Extensions for details.

Manifests

Overview

It is possible to install a component manually by copying the files using an SFTP client and modifying the database tables. It is more efficient to create a package file in the form on an XML document that will allow the Joomla! Installer to do this for you. This package file contains a variety of information:

- basic descriptive details about your component (i.e. name), and optionally, a description, copyright and license information.
- a list of files that need to be copied.
- optionally, a PHP file that performs additional install and uninstall operations.
- optionally, an SQL file which contains database queries that should be executed upon install/uninstall

Note: All components must be prefixed with com_.

Structure

This XML file just lines out basic information about the template such as the owner, version, etc. for identification by the Joomla! installer and then provides optional parameters which may be set in the Module Manager and accessed from within the module's logic to fine tune its behavior. Additionally, this file tells the installer which files should be copied and installed.

A typical component manifest:

```
<?xml version="1.0" encoding="utf-8"?>
<install type="component" version="1.5.0">
 <name>hello_world</name>
 <!-- The following elements are optional and free of formatting contt
raints -->
 <creationDate>2007 01 17</creationDate>
 <author>John Doe</author>
 <authorEmail>john.doe@example.org</authorEmail>
 <authorUrl>http://www.example.org</authorUrl>
 <copyright>Copyright Info</copyright>
 <license>License Info</license>
 <!--  The version string is recorded in the components table -->
 <version>Component Version String</version>
 <!-- The description is optional and defaults to the name -->
 <description>Description of the component ...</description>

 <!-- Custom Install Script to execute -->
```

```
 <!-- Note: This will be copied from the root of the installation pack
age to the administrator directory automatically -->
 <installfile>install.eventlist.php</installfile>

 <!-- Custom Uninstall Script to execute -->
 <!-- Note: This will be copied from the root of the installation pack
age to the administrator directory automatically -->
 <uninstallfile>uninstall.eventlist.php</uninstallfile>

 <!-- Install Database Section -->
 <install>
  <sql>
   <file driver="mysql" charset="utf8">install.mysql.utf8.sql</file>
   <file driver="mysql">install.mysql.nonutf8.sql</file>
  </sql>
 </install>

 <!-- Uninstall Database Section -->
 <uninstall>
  <sql>
   <file driver="mysql" charset="utf8">uninstall.mysql.utf8.sql</file>
   <file driver="mysql">uninstall.mysql.nonutf8.sql</file>
  </sql>
 </uninstall>

 <!-- Site Main File Copy Section -->
 <files>
  <filename>index.html</filename>
  <filename>test.php</filename>
  <folder>views</folder>
 </files>

 <!-- Site Main Language File Copy Section -->
 <languages>
  <language tag="en-GB">en-GB.com_test.ini</language>
  <language tag="de-DE">de-DE.com_test.ini</language>
  <language tag="nl-NL">nl-NL.com_test.ini</language>
 </languages>

 <!-- Site Main Media File Copy Section -->
 <media destination="com_test">
  <filename>image.png</filename>
  <filename>flash.swf</filename>
 </media>

 <administration>
```

```
  <!-- Administration Menu Section -->
  <menu img="components/com_test/assets/test-16.png">EventList</menu>
  <submenu>
   <!-- Note that all & must be escaped to & for the file to be valid
XML and be parsed by the installer -->
   <menu link="option=com_helloworld&task=hello&who=world">Hello World
!</menu>
   <!-- Instead of link you can specify individual link attributes -->
   <menu img="icon" task="hello" controller="z" view="a" layout="b" su
b="c">Hello Again!</menu>
   <menu view="test" layout="foo">Testing Foo Layout</menu>
  </submenu>

  <!-- Administration Main File Copy Section -->
  <!-- Note the folder attribute: This attribute describes the folder
    to copy FROM in the package to install therefore files copied
    in this section are copied from /admin/ in the package -->
  <files folder="admin">
   <filename>index.html</filename>
   <filename>admin.test.php</filename>
  </files>

  <!-- Administration Language File Copy Section -->
  <languages folder="admin">
   <language tag="en-GB">en-GB.com_test.ini</language>
   <language tag="de-DE">de-DE.com_test.ini</language>
   <language tag="nl-NL">nl-NL.com_test.ini</language>
  </languages>

  <!-- Administration Main Media File Copy Section -->
  <media folder="admin" destination="com_test">
   <filename>admin-image.png</filename>
   <filename>admin-flash.swf</filename>
  </media>
 </administration>
</install>
```

Entry Point

Overview

Joomla! 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:

```
index.php?option=com_hello&view=hello
```

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

Implementation

Joomla! 1.5 Methodology

The code for this file is fairly typical across components.

```
<?php
// No direct access
defined( '_JEXEC' ) or die( 'Restricted access' );

// Require the base controller
require_once( JPATH_COMPONENT.DS.'controller.php' );

// Require specific controller if requested
if ($controller = JRequest::getWord('controller')) {
 $path = JPATH_COMPONENT.DS.'controllers'.DS.$controller.'.php';
 if (file_exists($path)) {
  require_once $path;
 } else {
  $controller = '';
 }
}

// Create the controller
$classname = 'HelloController'.$controller;
$controller = new $classname( );

// Perform the Request task
```

```
$controller->execute( JRequest::getWord( 'task' ) );

// Redirect if set by the controller
$controller->redirect();
```

The first statement is a security check.

JPATH_COMPONENT is the absolute path to the current component, in our case components/com_hello. If you specifically need either the Site component or the Administrator component, you can use JPATH_COMPONENT_SITE or JPATH_COMPONENT_ADMINISTRATOR.

DS is 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 OSs. The 'DS' constant should always be used when referring to files on the local server.

After loading the base controller, we check if a specific controller is needed. In this component, the base controller is the only controller, but we will leave this conditional check "in place" for future use.

JRequest:getWord() finds a word variable in the URL or the POST data. So if our URL is index.php?option=com_hello&controller=controller_name, then we can retrieve our controller name in our component using: echo JRequest::getWord('controller');

Now we have our base controller 'HelloController' in com_hello/controller.php, and, if needed, additional controllers like 'HelloControllerController1' in com_hello/controllers/controller1.php. Using this standard naming scheme will make things easy later on: '{Componentname}{Controller}{Controllername}'

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 controller might decide to redirect the page, usually after a task like 'save' has been completed. This last statement takes care of the actual redirection.

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 since Joomla! 1.5,

and will be used for all php-only files.

HUBzero Methodology

HUBzero components differ in subtle, but key ways from standard Joomla! components. This is, in part, due to legacy issues. Some changes are made to aid in development while others may simply be a difference in philosophy. Note, however, that **no** differences require hacking or altering Joomla! in any way and HUBzero methodologies will run on any stock Joomla! install.

```
<?php
// No direct access
defined( '_JEXEC' ) or die( 'Restricted access' );

// Check if debugging is turned on
// If it is, we'll turn on PHP error reporting so we can more clearly
see our PHP bugs
$config = JFactory::getConfig();
if ($config->getValue('config.debug')) {
 error_reporting(E_ALL);
 @ini_set('display_errors','1');
}

// Include the JView class
jimport('joomla.application.component.view');

// Require the base controller
require_once( JPATH_COMPONENT.DS.'controller.php' );

// Require specific controller if requested
if ($controller = JRequest::getWord('controller')) {
 $path = JPATH_COMPONENT.DS.'controllers'.DS.$controller.'.php';
 if (file_exists($path)) {
  require_once $path;
 } else {
  $controller = '';
 }
}

// Create the controller
$classname = 'HelloController'.$controller;
$controller = new $classname( );

// Perform the Request task
$controller->execute();

// Redirect if set by the controller
```

```
$controller->redirect();
```

Here, you can see we've added a few things and made one subtle change in calling the execute() method. First, we added some lines that check if site debugging is turned on. If so, we turn on PHP error reporting. This can aid greatly in development.

Next, we added the jimport call to include the component JView class. This is done specifically because HUBzero controllers do **not** extend JController. JController does some autoloading and initial setup for Joomla! components and since we're not employing it, we need to do some class loading and setup of our own.

Finally, we removed the JRequest::getWord('task') being passed to the execute() method. HUBzero controllers handle the task request within the execute() method, rather than passing the task to it.

Controllers

Overview

The controller is responsible for responding to user actions. In the case of a web application, a user action is (generally) a page request. The controller will determine what request is being made by the user and respond appropriately by triggering the model to manipulate the data appropriately and passing the model into the view. The controller does not display the data in the model, it only triggers methods in the model which modify the data, and then pass the model into the view which displays the data.

Most components have two controllers: one for the front-end and one for the back-end.

Creating the Front-end Controller

Joomla! 1.5 Method

Our component only has one task - greet the world. Therefore, the controller will be very simple. No data manipulation is required. All that needs to be done is the appropriate view loaded. We will have only one method in our controller: display(). Most of the required functionality is built into the JController class, so all that we need to do is invoke the JController::display() method.

The code for the base controller com_hello/controller.php is:

```
<?php
// No direct access
defined( '_JEXEC' ) or die( 'Restricted access' );

jimport('joomla.application.component.controller');

/**
 * Hello World Component Controller
 *
 * @package    Joomla.Tutorials
 * @subpackage Components
 */
class HelloController extends JController
{
    /**
     * Method to display the view
     *
     * @access    public
     */
    public function display()
    {
```

```
        parent::display();
    }
}
```

The JController constructor will always register a display() task and unless otherwise specified (using the registerDefaultTask() method), it will set it as the default task.

This barebones display() method isn't really even necessary since all it does is invoke the parent constructor. However, it is a good visual clue to indicate what is happening in the controller.

The JController::display() method will determine the name of the view and layout from the request and load that view and set the layout. When you create a menu item for your component, the menu manager will allow the administrator to select the view that they would like the menu link to display and to specify the layout. A view usually refers to a view of a certain set of data (i.e. a list of cars, a list of events, a single car, a single event). A layout is a way that that view is organized.

In our component, we will have a single view called hello, and a single layout (default).

HUBzero Method

Most HUBzero component controllers will differ from Joomla! 1.5 in some important ways. This is, in part, due to legacy issues. Some changes are made to aid in development while others may simply be a difference in philosophy. Note, however, that no differences require hacking or altering Joomla! in any way and HUBzero methodologies will run on any stock Joomla! install.

```
<?php
// No direct access
defined('_JEXEC') or die('Restricted access');

class HelloController extends JObject
{
 private $_name  = NULL;
 private $_data  = array();
 private $_task  = NULL;

 //-----------

 public function __construct( $config=array() )
 {
  $this->_redirect = NULL;
  $this->_message = NULL;
```

```
  $this->_messageType = 'message';

  // Set the controller name
  if (empty( $this->_name )) {
   if (isset($config['name'])) {
    $this->_name = $config['name'];
   } else {
    $r = null;
    if (!preg_match('/(.*)Controller/i', get_class($this), $r)) {
     echo "Controller::__construct() : Can't get or parse class name."
;
    }
    $this->_name = strtolower( $r[1] );
   }
  }

  // Set the component name
  $this->_option = 'com_'.$this->_name;
 }

 //-----------

 public function __set($property, $value)
 {
  $this->_data[$property] = $value;
 }

 //-----------

 public function __get($property)
 {
  if (isset($this->_data[$property])) {
   return $this->_data[$property];
  }
 }

 //-----------

 public function execute()
 {
  $this->_task = JRequest::getString('task', '');

  switch ($this->_task)
  {
   default: $this->hello(); break;
  }
```

```
 }

 //-----------

 public function redirect()
 {
  if ($this->_redirect != NULL) {
   $app =& JFactory::getApplication();
   $app->redirect( $this->_redirect, $this->_message );
  }
 }

 //-----------

 protected function hello()
 {
  // Instantiate a new view
  $view = new JView( array('name'=>'hello') );

  // Pass the view any data it may need
  $view->greeting = 'Hello, World!';

  // Set any errors
  if ($this->getError()) {
   $view->setError( $this->getError() );
  }

  // Output the HTML
  $view->display();
 }
}
```

There appears to be a bit more going on here than in the Joomla! example but both code examples are doing essentially the same thing, as we'll explain.

The first, and most important, difference to note is that we're extending JObject rather than JController. Since we're not employing JController, we need to set up many of our methods manually. Much of it, such as the __constructor and redirect methods are very similar to the Joomla! method--they're simply being established here rather than in JController.

One key difference is how the execute() method is handled. In Joomla! 1.5, any public method is assumed to be an executable task. In the HUBzero method, we're explicitly declaring a list of available tasks and what those tasks execute via the switch statement.

Finally, in our display method, we're instantiating a new view, assigning it some data, and then displaying the output.

Helpers

Overview

A helper class is a class filled with static methods and is usually used to isolate a "useful" algorithm. They are used to assist in providing some functionality, though that functionality isn't the main goal of the application. They're also used to reduce the amount of redundancy in your code.

Implementation

Helper classes are stored in the helper sub-directory of your component directory. Naming convention typically follows a pattern of {ComponentName)Helper(HelperName}. Therefore, our helper class is called HelloHelperOutput.

Here's our com_hello/helpers/output.php helper class:

```
<?php
// No direct access

defined( '_JEXEC' ) or die( 'Restricted access' );

jimport('joomla.application.component.helper');

/**
 * Hello World Component Helper
 *
 * @package    Joomla.Tutorials
 * @subpackage Components
 */
class HelloHelperOutput
{
 /**
  * Method to make all text upper case
  *
   * @access    public
  */
 public function shout($txt='')
 {
  return strToUpper($txt).'!';
 }
}
```

We have one method in this class that takes all strings passed to it and returns them uppercase with an exclamation point attached to the end. To use this helper, we do the following:

```
class HelloViewHello extends JView
{
 function display($tpl = null)
 {
  include_once(JPATH_COMPONENT.DS.'helpers'.DS.'output.php');

  $greeting = HelloHelperOutput::shout("Hello World");
  $this->assignRef( 'greeting', $greeting );

  parent::display($tpl);
 }
}
```

Models

Overview

The concept of model gets its name because this class is intended to represent (or 'model') some entity.

Creating A Model

All Joomla! models extend the JModel class. The naming convention for models in the Joomla! framework is that the class name starts with the name of the component, followed by 'model', followed by the model name. Therefore, our model class is called HelloModelHello.

```
<?php
// No direct access
defined( '_JEXEC' ) or die( 'Restricted access' );

jimport( 'joomla.application.component.model' );

/**
 * Hello Model
 */
class HelloModelHello extends JModel
{
    /**
    * Gets the greeting
    * @return string The greeting to be displayed to the user
    */
    function getGreeting()
    {
        return 'Hello, World!';
    }
}
```

You will notice a line that starts with jimport. The jimport function is used to load files from the Joomla! framework that are required for our component. This particular statement will load the file /libraries/joomla/application/component/model.php. The '.'s are used as directory separators and the last part is the name of the file to load. All files are loaded relative to the libraries directory. This particular file contains the class definition for the JModel class, which is necessary because our model extends this class.

Using A Model

The Joomla! framework is setup in such a way that the controller will automatically load the model that has the same name as the view and will push it into the view. We can easily retrieve a reference to our model using the JView::getModel() method. If the model had not followed this convention, we could have passed the model name to JView::getModel().

Here's an example of using a model with our Hello component (com_hello).

```
<?php
// No direct access
defined( '_JEXEC' ) or die( 'Restricted access' );

jimport( 'joomla.application.component.view');

/**
 * HTML View class for the HelloWorld Component
 *
 * @package    HelloWorld
 */

class HelloViewHello extends JView
{
 function display($tpl = null)
 {
  $model = &$this->getModel();
  $greeting = $model->getGreeting();
  $this->assignRef( 'greeting', $greeting );

  parent::display($tpl);
 }
}
```

Languages

Setup

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

```
; Module - Hellow World (en-US)
COM_HELLOWORLD_LABEL_USER_COUNT = "User Count"
COM_HELLOWORLD_DESC_USER_COUNT = "The number of users to display"
COM_HELLOWORLD_RANDOM_USERS = "Random Users for Hello World"
COM_HELLOWORLD_USER_LABEL = "%s is a randomly selected user"
```

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 COM_{ComponentName}_{Text} for naming. Adhering to this naming convention is not required but is strongly recommended as it can help avoid potential translation collisions.

See the Languages overview for details.

Translating Text

Below is an example of accessing the translate helper:

```
<p><?php echo JText::_("COM_EXAMPLE_MY_LINE"); ?></p>
```

JText::_ is used for simple strings.
JText::sprintf is used for strings that require dynamic data passed to them for variable replacement.

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

See the Languages overview for details.

Layouts

Directory Structures & Files

Views are written in PHP and HTML and have a .php file extension. View scripts are placed in /com_{componentname}/views/, where they are further categorized by the /{viewname}/tmpl. Within these subdirectories, you will then find and create view scripts that correspond to each controller action exposed; in the default case, we have the view script default.php.

```
/hubzero
  /components
    /com_{componentname}
      /views
        /{viewname}
          view.html.php
          /tmpl
            default.php
```

Overriding module and component presentation in templates is further explained in the Templates: Overrides section.

Creating A View

Joomla! 1.5 Method

The task of the view is very simple: It retrieves the data to be displayed and pushes it into the template. Data is pushed into the template using the JView::assignRef method.

```
<?php

// no direct access
defined( '_JEXEC' ) or die( 'Restricted access' );

jimport( 'joomla.application.component.view');

/**
 * HTML View class for the HelloWorld Component
 *
 * @package    HelloWorld
 */

class HelloViewHello extends JView
```

```
{
    function display($tpl = null)
    {
        $greeting = "Hello World!";
        $this->assignRef( 'greeting', $greeting );

        parent::display($tpl);
    }
}
```

HUBzero Method

Not necessary. Data retrieval and template assignment is handled in the controller.

Creating the Template

Joomla! templates/layouts are regular PHP files that are used to layout the data from the view in a particular manner. The variables assigned by the JView::assignRef method can be accessed from the template using $this->{propertyname} (see the template code below for an example).

Our template is very simple: we only want to display the greeting that was passed in from the view - this file is: views/hello/tmpl/default.php:

```
<?php

// No direct access
defined('_JEXEC') or die('Restricted access'); ?>
<h1><?php echo $this->greeting; ?></h1>
```

Routing

Overview

All components in Joomla! can be accessed through a query string by using the option parameter which will equate to the name of the component. For example, to access the "Contacts" component, you could type http://yourhub.org/index.php?option=com_contact.

When SEF URLs are being employed, the first portion after the site name will almost always be the name of a component. For the URL http://yourhub.org/contact, the first portion after the slash translates to the component com_contact. If a matching component cannot be found, routing will attempt to match against an article section, category, and/or page alias.

While not required, most components will have more detailed routing instructions that allow SEF URLs to be made from and converted back into query strings that pass necessary data to the component. This is done by the inclusion of a file called router.php.

router.php

Every router.php file has two methods: {ComponentName}BuildRoute() which takes a query string and turns it into a SEF URL and {ComponentName}ParseRoute() which deconstructs a SEF URL back into a query string to be passed to the component.

```
function ExampleBuildRoute(&$query)
{
 $segments = array();

 if (!empty($query['task'])) {
  $segments[] = $query['task'];
  unset($query['task']);
 }
 if (!empty($query['id'])) {
  $segments[] = $query['id'];
  unset($query['id']);
 }
 if (!empty($query['format'])) {
  $segments[] = $query['format'];
  unset($query['format']);
 }

 return $segments;
}

function ExampleParseRoute($segments)
```

```
{
 $vars = array();

 if (empty($segments)) {
  return $vars;
 }
 if (isset($segments[0])) {
  $vars['task'] = $segments[0];
 }
 if (isset($segments[1])) {
  $vars['id'] = $segments[1];
 }
 if (isset($segments[2])) {
  $vars['format'] = $segments[2];
 }

    return $vars;
}
```

{ComponentName}BuildRoute()

This method is called when using JRoute::_(). JRoute::_() passes the query string (minus the option={componentname} portion) to the method which returns an array containing the necessary portions of the URL to be constructed *in the order* they need to appear in the final SEF URL.

```
// $query = 'task=view&id=123&format=rss'
function ExampleBuildRoute(&$query)
{
 $segments = array();

 if (!empty($query['task'])) {
  $segments[] = $query['task'];
  unset($query['task']);
 }
 if (!empty($query['id'])) {
  $segments[] = $query['id'];
  unset($query['id']);
 }
 if (!empty($query['format'])) {
  $segments[] = $query['format'];
  unset($query['format']);
 }
```

```
 return $segments;
}
```

Will return:

```
Array(
 'view',
 '123',
 'rss'
);
```

This will in turn be passed back to JRoute::_() which will construct the final SEF URL of example/view/123/rss.

{ComponentName}ParseRoute()

This method is automatically called on each page view. It is passed an array of segments of the SEF URL that called the page. That is, a URL of example/view/123/rss would be separated by the forward slashes with the first segment automatically being associated with a component name. The rest are stored in an array and passed to {ComponentName}ParseRoute() which then associates each segment with an appropriate variable name based on the segment's position in the array.

```
function ExampleParseRoute($segments)
{
 $vars = array();

 if (empty($segments)) {
  return $vars;
 }
 if (isset($segments[0])) {
  $vars['task'] = $segments[0];
 }
 if (isset($segments[1])) {
  $vars['id'] = $segments[1];
 }
 if (isset($segments[2])) {
  $vars['format'] = $segments[2];
 }
```

```
    return $vars;
}
```

Note: Position of segments is very important here. A URL of example/view/123/rss could yield completely different results than a URL of example/rss/view/123.

Packaging

Overview

Packaging a component for distribution is relatively easy. All front-end files are places within a directory called /site and all administration files are placed within a directory called /admin. Here's what a typical package will look like:

```
/com_{componentname}
  {componentname}.xml
  /site
    {componentname}.php
    controller.php
    /views
      /{viewname}
        view.html.php
        /tmpl
          default.php
    /models
      {modelname}.php
    /controllers
      {controllername}.php
  /admin
    {componentname}.php
    controller.php
    /views
      /{viewname}
        view.html.php
        /tmpl
          default.php
    /models
      {modelname}.php
    /controllers
      {controllername}.php
```

Just "zip" up the primary directory into a compressed archive file. When the ZIP file is installed, the language file is copied to /language/{LanguageName}/{LanguageName}.{ComponentName}.ini and is loaded each time the module is loaded. All of the other files are copied to the /components/{ComponentName} and /administrator/components/{ComponentName}directories of the Joomla! installation.

Modules

Overview

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 module 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.

Directory Structure & Files

The directory structure used allows you to separate different MVC applications into self-contained units. This helps keep related code organized, easy to find, and can make redistribution as packages considerably easier. To illustrate the typical module directory structure and files:

```
/hubzero
  /modules
    /mod_{ModuleName}
      /tmpl
        default.php
      helper.php
      mod_{ModuleName}.php
      mod_{ModuleName}.xml
```

A Joomla! 1.5 Module is in its most basic form two files: an XML configuration file and a PHP controller file. Typically, however, a module will also include a view file which contains the HTML and presentation aspects.

/tmpl
: This directory contains template files.
: default.php
: : This is the module template. This file will take the data collected by mod_{ModuleName}.php and generate the HTML to be displayed on the page.

helper.php
: This file contains the helper class which is used to do the actual work in retrieving the information to be displayed in the module (usually from the database or some other source).

mod_{ModuleName}.php
: This file is the main entry point for the module. It will perform any necessary initialization routines, call helper routines to collect any necessary data, and include the template which will display the module output.

mod_{ModuleName}.xml
: The XML configuration file contains general information about the module (as will be displayed in the Module Manager in the Joomla! administration interface), as well as module parameters which may be supplied to fine tune the appearance / functionality of the module.

While there is no restriction on the name itself, all modules must be prefixed with "mod_".

Examples

A simple "Hello, World" module:

Download: Hello World module (.zip)

A module demonstrating database access and language file:

Download: List Names module (.zip)

Installation

Installing

See Installing Extensions for details.

Uninstalling

See Uninstalling Extensions for details.

Manifests

Overview

All modules should include a manifest in the form of an XML document named the same as the module. The file holds key "metadata" about the module.

Note: All modules must be prefixed with mod_.

Directory Structure & Files

Manifests are stored in the same directory as the module file itself and must be named the same (the file extension being the obvious difference).

```
/hubzero
  /modules
    /{ModuleName}
      /tmpl
        default.php
      helper.php
      mod_{ModuleName}.php
      mod_{ModuleName}.xml
```

Structure

This XML file just lines out basic information about the template such as the owner, version, etc. for identification by the Joomla! installer and then provides optional parameters which may be set in the Module Manager and accessed from within the module's logic to fine tune its behavior. Additionally, this file tells the installer which files should be copied and installed.

A typical module manifest:

```
<?xml version="1.0" encoding="utf-8"?>
<install type="module" version="1.5.0">
 <!-- Name of the Module -->
 <name>Hello World - Hello</name>

 <!-- Name of the Author -->
 <author>Ambitionality Software LLC</author>

 <!-- Version Date of the Module -->
 <creationDate>2008-06-23</creationDate>
```

```
 <!-- Copyright information -->
 <copyright>All rights reserved by Ambitionality Software LLC 2008.</c
opyright>

 <!-- License Information -->
 <license>GPL 2.0</license>

 <!-- Author's email address -->
 <authorEmail>info@ambitionality.com</authorEmail>

 <!-- Author's website -->
 <authorUrl>www.ambitionality.com</authorUrl>

 <!-- Module version number -->
 <version>1.0.0</version>

 <!-- Description of what the module does -->
 <description>Outputs a random list of user names</description>

 <!-- Listing of all files that should be installed for the module to
function -->
 <files>
  <!-- The "module" attribute signifies that this is the main controll
er file -->
  <filename module="mod_listnames">mod_listnames.php</filename>
  <filename>index.html</filename>
  <filename>tmpl/default.php</filename>
  <filename>tmpl/index.html</filename>
 </files>

 <languages>
  <!-- Any language files included with the module -->
  <language tag="en-GB">en-GB.mod_listnames.ini</language>
 </languages>

 <!-- Optional parameters -->
 <params>
  <!-- parameter to allow placement of a module class suffix for the m
odule table / xhtml display -->
  <param name="moduleclass_sfx" type="text" default="" label="Module C
lass Suffix" description="PARAMMODULECLASSSUFFIX" />

  <!-- just gives us a little room between the previous paramter and t
he next -->
  <param name="@spacer" type="spacer" default="" label="" description=
```

```
"" />

  <!-- A parameter that allows an administrator to modify the number o
f users that this module will display -->
  <param name="usercount" type="text" default="5" label="LABEL USER CO
UNT" description="DESC USER COUNT" />
 </params>
</install>
```

Note: Notice that we DO NOT include a reference in the files section for the XML file.

Let's go through some of the most important tags:

INSTALL
: The install tag has several key attributes. The type must be "module".

NAME
: You can name the module in any way you wish.

FILES
: The files tag includes all of the files that will will be installed with the module.

PARAMS
: Any number of parameters can be specified for a module.

See Joomla!'s Documentation on the full list of available parameter types and what they do.

Controllers

Overview

Unlike components, which potentially can have multiple controllers, modules do not have a controller class. As such, the module directory structure doesn't include a /controllers subdirectory or controller.php. Instead, the setting of parameters, inclusion of any necessary files, and the instantiation of the module's view are done within the mod_{ModuleName}.php file.

Directory Structure & Files

The controller is stored in the same directory as the module file itself and must be named the same (the file extension being the obvious difference).

```
/hubzero
  /modules
    /{ModuleName}
      /tmpl
        default.php
      helper.php
      mod_{ModuleName}.php
      mod_{ModuleName}.xml
```

Implementation

Most modules will perform three tasks in the following order:

- Include the helper.php file which contains the class to be used to collect any necessary data
- Invoke the appropriate helper class method to retrieve any data that needs to be available to the view
- Include the template to display the output

Here are the contents of mod_listnames.php:

```
<?php
// No direct access
defined('_JEXEC') or die('Restricted access');

// Include the helper file
require_once(dirname(__FILE__).DS.'helper.php');
```

```
// Get a parameter from the module's configuration
$userCount = $params->get('usercount');

// Get the items to display from the helper
$items = modListNamesHelper::getItems($userCount);

// Include the template for display
require(JModuleHelper::getLayoutPath('mod_listnames'));
```

Helpers

Overview

The helper.php file contains that helper class that is used to retrieve the data to be displayed in the module output. Most modules will have at least one helper but it is possible to have a module with more or none.

Directory Structure & Files

The directory structure used for MVC oriented modules includes the helper.php file in the top directory for that module. While there is no rule stating that we must name our helper class as we have, but it is helpful to do this so that it is easily identifiable and locateable.

```
/hubzero
  /modules
    /mod_{ModuleName}
      helper.php
```

Implementation

In our mod_helloworld example, the helper class will have one method: getItems(). This method will return the items we retrieved from the database.

Here is the code for the mod_helloworld helper.php file:

```
<?php
// No direct access
defined('_JEXEC') or die('Restricted access');

class modHelloWorldHelper
{
    /**
     * Retrieves the hello message
     *
     * @param array $params An object containing the module parameters
     * @access public
     */
    public function getItems( $userCount )
    {
        return 'Hello, World!';
    }
```

```
}
```

More advanced modules might include multiple database requests or other functionality in the helper class method.

Languages

Setup

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

```
; Module - List Names (en-US)
MOD_LISTNAMES_LABEL_USER_COUNT = "User Count"
MOD_LISTNAMES_DESC_USER_COUNT = "The number of users to display"
MOD_LISTNAMES_RANDOM_USERS = "Random Users for Hello World"
MOD_LISTNAMES_USER_LABEL = "%s is a randomly selected user"
```

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 MOD_{ModuleName}_{Text} for naming. Adhering to this naming convention is not required but is strongly recommended as it can help avoid potential translation collisions.

See the Languages overview for details.

Translating Text

Below is an example of accessing the translate helper:

```
<p><?php echo JText::_("MOD_EXAMPLE_MY_LINE"); ?></p>
```

JText::_ is used for simple strings.
JText::sprintf is used for strings that require dynamic data passed to them for variable replacement.

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

See the Languages overview for details.

Layouts

Overview

While technically not necessary for a module to function, it is considered best practices to have a more MVC structure to your module and put all HTML and display code into view files. This allows for separation of the logic from presentation. There is a second advantage to this, however, which is that it will allow the presentation to be overridden easily by any Joomla! 1.5 template for optimal integration into any site.

Overriding module and component presentation in templates is further explained in the Templates: Overrides section.

Directory Structure & Files

The directory structure used for MVC oriented modules includes a tmpl directory for storing view files. While more views may be possible, modules should include at least one view names default.php.

```
/hubzero
  /modules
    /mod_{ModuleName}
      /tmpl
        default.php
```

Implementation

A simple view (default.php) for a module named mod_listnames:

```
<?php defined('_JEXEC') or die('Restricted access'); // no direct acce
ss ?>
<?php echo JText::_('MOD_LISTNAMES_RANDOM_USERS'); ?>
<ul>
 <?php foreach ($items as $item) { ?>
 <li>
  <?php echo JText::sprintf('MOD_LISTNAMES_USER_LABEL', $item->name);
?>
 </li>
 <?php } ?>
</ul>
```

Here we simply create an unordered HTML list and then iterate through the items returned by our helper (in mod_listnames.php), printing out a message with each user's name.

An important point to note is that the template file has the same scope as the mod_listnames.php file. What this means is that the variable $items can be defined in the mod_listnames.php file and then used in the default.php file without any extra declarations or function calls.

Now that we have a view to display our data, we need to tell the module to load it. This is done in the module's controller file and typically occurs last.

```
<?php
// No direct access
defined('_JEXEC') or die('Restricted access');

// Include the helper file
require_once(dirname(__FILE__).DS.'helper.php');

// Get a parameter from the module's configuration
$userCount = $params->get('usercount');

// Get the items to display from the helper
$items = modListNamesHelper::getItems($userCount);

// Include the template for display
require(JModuleHelper::getLayoutPath('mod_listnames'));
```

Here we can see that the name of the module must be passed to the getLayoutPath method of JModuleHelper. This will load default.php and stores the output in an output buffer which is then rendered onto the page output.

Packaging

Overview

Packaging a module for distribution is easy. Just "zip" up the module directory into a compressed archive file. When the ZIP file is installed, the language file is copied to /language/{LanguageName}/{LanguageName}.{ModuleName}.ini and is loaded each time the module is loaded. All of the other files are copied to the /modules/{ModuleName} subfolder of the Joomla! installation.

Loading

Loading in Templates

Modules may be loaded in a template by including a Joomla! specific jdoc:include tag. This tag includes two attributes: type, which must be specified as module in this case and name, which specifies the position that you wish to load. Any modules assigned to the specified position (set via the administrative Module Manager) declared in the name attribute will have their output placed in the template (the jdoc:include is removed by Joomla! afterwards).

```
<jdoc:include type="modules" name="footer" />
```

Advanced Template Loading

The countModules method can be used within a template to determine the number of modules enabled in a given module position. This is commonly used to include HTML around modules in a certain position only if at least one module is enabled for that position. This prevents empty regions from being defined in the template output and is a technique sometimes referred to as "collapsing columns".

For example, the following code includes modules in the 'user1' position only if at least one module is enabled for that position.

```
<?php if ($this->countModules( 'user1' )) : ?>
 <div class="user1">
  <jdoc:include type="modules" name="user1" />
 </div>
<?php endif; ?>
```

The countModules method can be used to determine the number of Modules in more than one Module position. More advanced calculations can also be performed.

The argument to the countModules function is normally just the name of a single Module position. The function will return the number of Modules currently enabled for that Module position. But you can also do simple logical and arithmetic operations on two or more Module positions.

```
$this->countModules( 'user1 + user2' );
```

Although the usual arithmetic operators, +. -. *, / will work as expected, these are not as useful as the logical operators 'and' and 'or'. For example, to determine if the 'user1' position and the 'user2' position both have at least one Module enabled, you can use the function call:

```
$this->countModules( 'user1 and user2' );
```

Careful: A common mistake is to try something like this:

```
$this->countModules( 'user1' and 'user2' );
```

This will return false regardless of the number of Modules enabled in either position, so check what you are passing to countModules carefully.

You must have exactly one space character separating each item in the string. For example, 'user1+user2' will not produce the desired result as there must be a space character either side of the '+' sign. Also, 'user1 &nbp;+ user2' will produce an error message as there is more than one space separating each element.

Example using the or operator: The user1 and user2 Module positions are to be displayed in the region, but you want the region to not appear at all if no Modules are enabled in either position.

```
<?php if ($this->countModules( 'user1 or user2' )) : ?>
 <div class="rightcolumn">
  <jdoc:include type="modules" name="user1" />
  <jdoc:include type="modules" name="user2" />
 </div>
<?php endif; ?>
```

Advanced example: The user1 and user2 Module positions are to be displayed side-by-side with a separator between them. However, if only one of the Module positions has any Modules enabled then the separator is not needed. Furthermore, if neither user1 or user2 has any Modules enabled then nothing is output.

```
<?php if ($this->countModules( 'user1 or user2' )) : ?>
 <div class="user1user2">

 <?php if ($this->countModules( 'user1' )) : ?>
```

```
  <jdoc:include type="modules" name="user1" style="xhtml" />
 <?php endif; ?>

 <?php if ($this->countModules( 'user1 and user2' )) : ?>
  <div class="greyline"></div>
 <?php endif; ?>

 <?php if ($this->countModules( 'user2' )) : ?>
  <jdoc:include type="modules" name="user2" style="xhtml" />
 <?php endif; ?>

 </div>
<?php endif; ?>
```

Notice how the first countModules call determines if there any Modules to display at all. The second determines if there are any in the 'user1' position and if there are it displays them. The third call determines if both 'user1' and 'user2' positions have any Modules enabled and if they do then if provides a separator between them. Finally, the fourth call determines if there are any enabled Modules in the 'user2' position and displays them if there are any.

Loading in Components

Sometimes it is necessary to render a module within a component. This can be done with the XModuleHelper class provided by HUBzero. To import the class, you must first use the ximport('name of file or class') method.

XModuleHelper::renderModules($position)

> Used for loading potentially multiple modules assigned to a position. This will capture the rendered output of all modules assigned to the $position parameter passed to it and return the compiled output.
>
> ```
> ximport('xmodule');
> $output = XModuleHelper::renderModules('footer');
> ```

XModuleHelper::renderModule($name)

> Used for loading a single module of a specific name. This will capture the rendered output of the module with the $name parameter passed to it and return the compiled

output.

```
ximport('xmodule');
$output = XModuleHelper::renderModule('mod_footer');
```

XModuleHelper::displayModules($position)

Used for loading a single module of a specific name. This will echo rendered output of the module with the $name parameter passed to it.

```
ximport('xmodule');
XModuleHelper::displayModules('footer');
```

XModuleHelper::renderModule($name)

Used for loading a single module of a specific name. This will output the module with the $name parameter passed to it.

```
ximport('xmodule');
XModuleHelper::displayModule('mod_footer');
```

Loading in Articles

Modules may be loaded in an article by including a specific {xhub:module} tag. This tag includes one required attribute: position, which specifies the position that you wish to load. Any modules assigned to the specified position (set via the administrative Module Manager) declared in the position attribute will have their output placed in the article in the location of the {xhub:module} tag.

```
{xhub:module position="footer"}
```

Note: To use this feature, the xHUB Tags plugin for content must be installed and active.

Plugins

Overview

Joomla! 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. Joomla! plugins enable you to execute code in response to certain events, either Joomla! core events or custom events that are triggered from your own code. This is a powerful way of extending the basic Joomla! functionality.

See Joomla Events for a list of core Joomla plugin events.

See Component Events for a list of Hubzero plugin events.

Core Types

Plug-ins are managed at a group level that is defined in the plug-in's XML manifest file. While the number of possible types of plugins is almost limitless, there are a number of core plugin types that are used by Joomla!. These core types are grouped into directories under /plugins. They are:

- authentication
- content
- editors
- editors-xtd
- search
- system
- user
- xmlrpc

Authentication
: plugins allow you to authenticate (to allow you to login) against different sources. By default you will authenticate against the Joomla! user database when you try to login. However, there are other methods available such as by OpenID, by a Google account, LDAP, and many others. Wherever a source has a public API, you can write an authentication plugin to verify the login credentials against this source. For example, you could write a plugin to authenticate against Twitter accounts because they have a public API.

Content
: plugins modify and add features to displayed content. For example, content plugins can cloak email address or can convert URL's into SEF format. Content plugins can also look for markers in content and replace them with other text or HTML. For example, the Load Module plugin will take {*loadmodule banner1*} (you would remove the *'s in practice. They are included to actually prevent the plugin from working in this article),

load all the modules in the banner1 position and replace the marker with that output.

Editor
: plugins allow you to add new content editors (usually WYSIYWG).

Editor-XTD
: (extended) plugins allow you to add additional buttons to the editors. For example, the *Image*, *Pagebreak* and *Read more* buttons below the default editor are actually plugins.

Search
: plugins allow you to search different content from different components. For example, search plugins for Articles, Contacts and Weblinks are already provided in Joomla!.

System
: plugins allow you to perform actions at various points in the execution of the PHP code that runs a Joomla! Web site.

User
: plugins allow you to perform actions at different times with respect to users. Such times include logging in and out and also saving a user. User plugins are typically user to "bridge" between web applications (such as creating a Joomla! to phpBB bridge).

XML-RPC
: plugins allow you to provide additional XML-RPC web services for your site. When your Web site exposes web services, it gives you the ability to interact remotely, possibly from a desktop application. Web services are a fairly advanced topic and will not be covered in much detail here.

Directory & File Structure

While a plugin can have any number of files, there are two you need as a minimum and there are specific naming conventions you must follow. Before we look at the files, we must decide what sort of plugin we are going to create. It must either fall under one of the built-in types (authentication, content, editors, editors-xtd, search, system, user or xmlrpc) or your can create your own type by adding a new folder under /plugins. So, files for an authentication plugin will be saved under /plugins/authentication, files for a system plugin will be saved under /plugins/system, and so on.

The typical plugin install location and files:

```
/hubzero
  /plugins
    /{PluginType}
      {PluginName}.php
      {PluginName}.xml
```

As mentioned, a plugin has a minimum of two files: a PHP file, test.php, which is the file actually loaded by Joomla! and an XML file, text.xml, which contains meta and installation information for the plugin as well as the definition of the plugin parameters.

There is no restriction on the file name for the plugin (although we recommend sticking with alpha-numeric characters and underscores only), but once you decide on the file name, it will set the naming convention for other parts of the plugin.

Examples

A plugin demonstrating basic setup:

Download: System Test plugin (.zip)

Installation

Installing

See Installing Extensions for details.

Uninstalling

See Uninstalling Extensions for details.

Manifests

Overview

All plugins should include a manifest in the form of an XML document named the same as the plugin. So, a plugin named test.php would have an accompanying test.xml manifest.

Directory & Files

Manifests are stored in the same directory as the plugin file itself and must be named the same (file extension being the obvious exception).

```
/hubzero
  /plugins
    /{PluginType}
      {PluginName}.php
      {PluginName}.xml
```

Structure

A typical plugin manifest:

```
<?xml version="1.0" encoding="utf-8"?>
<install version="1.5.2" type="plugin" group="system" method="upgrade"
>
 <name>System - Test</name>
 <author>Author</author>
 <creationDate>Month 2008</creationDate>
 <copyright>Copyright (C) 2008 Holder. All rights reserved.</copyright
>
 <license>GNU General Public License</license>
 <authorEmail>email</authorEmail>
 <authorUrl>url</authorUrl>
 <version>1.0.1</version>
 <description>A test system plugin</description>
 <files>
  <filename plugin="example">example.php</filename>
 </files>
 <params>
    <param name="example"
     type="text"
     default=""
```

```
      label="Example"
      description="An example text parameter" />
  </params>
</install>
```

Let's go through some of the most important tags:

INSTALL
: The install tag has several key attributes. The type must be "plugin" and you must specify the group. The group attribute is required and is the name of the directory you saved your files in (for example, system, content, etc). We use the method="upgrade" attribute to allow us to install the extension without uninstalling. In other words, if you are sharing this plugin with other, they can just install the new version over the top of the old one.

NAME
: We usually start the name with the type of plugin this is. Our example is a system plugin and it has some some nebulous test purpose. So we have named the plugin "System - Test". You can name the plugins in any way, but this is a common format.

FILES
: The files tag includes all of the files that will will be installed with the plugin. Plugins can also support be installed with subdirectories. To specify these just all a FOLDER tag, <folder>test</folder>. It is common practice to have only one subdirectory and name it the same as the plugin PHP file (without the extension of course).

PARAMS
: Any number of parameters can be specified for a plugin. Please note there is no "advanced" group for plugins as there is in modules and components.

See Joomla!'s Documentation on the full list of available parameter types and what they do.

Controllers

Overview

All plugins will have a primary class extending JPlugin that contains the logic and events to be triggered.

Directory & Files

Plugin files are stored in a sub-directory of the /plugins directory. The sub-directory represents what type the plugin belongs to. This allows for plugins of the same name but for different types. For example, one could have a plugin named example for both the /system and /search types.

Note: plugins will always be within a type sub-directory and will never be found in the top-level /plugins directory.

```
/hubzero
  /plugins
    /{PluginType}
      {PluginName}.php
      {PluginName}.xml
```

There is no restriction on the file name for the plugin (although it is recommended to stick with alpha-numeric characters and underscores only), but once you decide on the file name, it will set the naming convention for other parts of the plugin.

Structure

Here we have a typical plugin class:

```
<?php
// no direct access
defined( '_JEXEC' ) or die( 'Restricted access' );

jimport( 'joomla.plugin.plugin' );

/**
 * Example system plugin
 */
class plgSystemTest extends JPlugin
```

```
{
 /**
  * Constructor
  *
  * For php4 compatibility we must not use the __constructor as a cons
tructor for plugins
  * because func_get_args ( void ) returns a copy of all passed argume
nts NOT references.
  * This causes problems with cross-
referencing necessary for the observer design pattern.
  *
  * @access protected
  * @param object $subject The object to observe
  * @param  array   $config  An array that holds the plugin configurat
ion
  * @since 1.0
  */
 function plgSystemTest( &$subject, $config )
 {
  parent::__construct( $subject, $config );

  // Do some extra initialization in this constructor if required
 }

 /**
  * Do something onAfterInitialise
  */
 function onAfterInitialise()
 {
  // Perform some action
 }
}
```

Let's look at this file in detail. Please note that the usual Docblock (the comment block you normally see at the top of most PHP files) has been omitted for clarity.

The file starts with the normal check for defined('_JEXEC') which ensures that the file will fail to execute if access directly via the URL. This is a very important security feature and the line must be placed before any other executable PHP in the file (it's fine to go after all the initial comment though). The importance of having this check your PHP files cannot be over-emphasised.

Next we use the jimport function to load the library file with the definition of the JPlugin class.

You will notice that a plugin is simply a class derived from JPlugin (this differs from previous versions of Joomla!). The naming convention of this class is very important. The formula for this name is:

plg + Proper case name of the plugin directory + Proper case name of the plugin file without the extension.

Proper case simply means that we capitalise the first letter of the name. When we join them altogether it's then referred to as "Camel Case". The case is not that important as PHP classes are not case-sensitive but it's the convention Joomla! uses and generally makes the code a little more readable.

For our test system plugin, the formula gives us a class name of:

plg + **S**ystem + **T**est = plgSystemTest

Let's move on to the methods in the class.

The first method, which is called the constructor, is completely optional. You only require this is you want to do some work when the plugin is actually loaded by Joomla!. This happens with a call to the helper method JPluginHelper::importPlugin(*<plugin_type>*). This means that you even if the plugin is never triggered, for whatever reason, you still have an opportunity to execute code if you need to in the constructor.

In PHP 4 the name of the constructor method is the same as the name of the class. If you were designing only for PHP 5 you could replace this with the name of __constructor instead.

The remaining methods will take on the name of "events" that are trigger throughout the execution of the Joomla! code. In the example, we know there is an event called onAfterInitialise which is the first event called after the Joomla! application sets itself up for work. For more information on when some events are triggered, see the API Execution Order page on the Documentation Wiki.

The naming rule here is simple: the name of the method must be the same as the event on which you want it triggered. The Joomla! Framework will auto-register all the methods in the class for you.

That's the basics of the plugin PHP file. It's location, name and methods will depend on what you want to use the plugin for.

Joomla Events

One thing to note about system plugins is that they are not limited to handling just system events. Because the system plugins are always loaded on each run of the Joomla! PHP, you can include any triggered event in a system plugin.

The events triggered in Joomla! are:

Authentication

- onAuthenticate

Content

- onPrepareContent
- onAfterDisplayTitle
- onBeforeDisplayContent
- onBeforeContentSave (new in 1.5.4)
- onAfterContentSave (new in 1.5.4)

Editors

- onInit
- onGetContent
- onSetContent
- onSave
- onDisplay
- onGetInsertMethod

Editors XTD (Extended)

- onDisplay

Search

- onSearch
- onSearchAreas

System

- onAfterInitialise
- onAfterRoute
- onAfterDispatch
- onAfterRender

User

- onLoginUser
- onLoginFailure
- onLogoutUser
- onLogoutFailure
- onBeforeStoreUser

- onAfterStoreUser
- onBeforeDeleteUser
- onAfterDeleteUser

XML-RPC

- onGetWebServices

For more detailed information on how to create specific plugins, visit the Plugins Category on the Joomla! Documentation Wiki.

Component Events

The following are events that are triggered from within their respective components:

Groups

- onGroupAreas
- onGroup
- onGroupNew
- onGroupDeleteCount
- onGroupDelete

Members

- onMembersAreas
- onMember

Tools

- onBeforeSessionInvoke
- onAfterSessionInvoke
- onBeforeSessionStart
- onAfterSessionStart
- onBeforeSessionStop
- onAfterSessionStop

Resources

- onResourcesAreas
- onResources

Support

- getReportedItem

- deleteReportedItem

Tags

- onTagAreas
- onTagView

Usage

- onUsageAreas
- onUsageDisplay

What's New

- onWhatsnewAreas
- onWhatsnew

XMessage

- onTakeAction
- onSendMessage
- onMessageMethods
- onMessage

XSearch

- onXSearchAreas
- onXSearch

Languages

Overview

Language translation files are placed inside the appropriate language languages directory within a widget.

```
/hubzero
  /language
    /{LanguageName}
      {LanguageName}.plg_{GroupName}_{PluginName}.ini
```

Note: Plugin language files contain data for both the front-end and administrative back-end.

Setup

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

```
; Plugin - System - Test (en-US)
PLG_SYSTEM_TEST_HERE_IS_LINE_ONE = "Here is line one"
PLG_SYSTEM_TEST_HERE_IS_LINE_TWO = "Here is line two"
PLG_SYSTEM_TEST_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 PLG_{PluginGroup}_{PluginName}_{Text} for naming. Adhering to this naming convention is not required but is strongly recommended as it can help avoid potential translation collisions.

See the Languages overview for details.

Loading

The appropriate language file for a plugin is **not** preloaded when the plugin is instantiated as many plugins may not have language files at all. As such, one must specifically load any file(s) if

they are needed. This can be done in the plugin's constructor but is more commonly found outside of the class altogether. Here we see the test plugin for the examples plugins group loading its language file right before declaration of the plugin's class.

```
<?php
// Check to ensure this file is included in Joomla!
defined('_JEXEC') or die( 'Restricted access' );

jimport( 'joomla.plugin.plugin' );
JPlugin::loadLanguage( 'plg_system_test' );

class plgSystemTest extends JPlugin
{
 ....
}
```

Note that the string passed to the loadLanguage() method matches the pattern for the naming of the language file itself, minus the language prefix and file extension.

Translating Text

Below is an example of accessing the translate helper:

```
<p><?php echo JText::_("PLGN_EXAMPLE_MY_LINE"); ?></p>
```

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

See the Languages overview for details.

Layouts

Overview

Note: Plugin views are an additional feature brought through HUBzero libraries. A standard, non-HUBzero Joomla! install will not have this capability.

The majority of plugins will not have view files. Occasionally, however, a plugin will return HTML and it is considered best practices to have a more MVC structure to your plugin and put all HTML and display code into view files. This allows for separation of the logic from presentation. There is a second advantage to this, however, which is that it will allow the presentation to be overridden easily by any Joomla! 1.5 template for optimal integration into any site.

Overriding plugin, module, and component presentation in templates is further explained in the Templates: Overrides section.

Directory Structure & Files

Plugins, like components and modules, are set up in a particular directory structure.

```
/plugins
  /groups
    forum.php   (the main plugin file)
    forum.xml   (the installation XML file)
    /forum
      /views
        /browse
          /tmpl
            default.php   (the layout)
            default.xml   (the layout installation XML file)
```

Similar to components, under the views directory of the plugin's self-titled directory (in the example, forum) there are directories for each view name. Within each view directory is a /tmpl/ directory. There is usually only one layout file but depending on who wrote the plugin, and how it is written, there could be more.

Implementation

Loading a plugin view

```
class plgExamplesTest extends JPlugin
{
    ...

    public function onReturnHtml()
    {
        // Include the HUBzero library that allows plugin views to wor
k
  ximport('Hubzero_Plugin_View');

  // Instantiate a new view
  $view = new Hubzero_Plugin_View(
   array(
    'folder'=>'examples',
    'element'=>'test',
    'name'=>'display'
   )
  );

  // Set any data the view may need
  $view->hello = 'Hello, World';

  // Set any errors
  if ($this->getError()) {
   $view->setError( $this->getError() );
  }

  // Return the view
  return $view->loadTemplate();
    }
}
```

In the example, we're instantiating a new plugin view and passing it an array of variables that tell the object where to load the view HTML from. folder is the plugin group, element is the plugin, and name is the name of the view that is to be loaded. So, in this case, it would correspond to a view found here:

```
/plugins
  /examples
    /test
      /views
        /display
          /tmpl
```

```
            default.php   (the layout)
            default.xml   (the layout installation XML file)
```

Also note that we're returning $view->loadTemplate() rather than calling $view->display(). The loadTemplate() method captures the HTML output of the view rather than printing it out to the screen. This allows us to store the output in a variable and pass it around for later display.

The plugin view file

Our view (default.php) is constructed the same as any module or component view file:

```
<?php defined('_JEXEC') or die('Restricted access'); // no direct acce
ss ?>
<p>
 <?php echo $this->hello; ?>
</p>
```

Packaging

Overview

Packaging a plugin for distribution is easy. If you only have the two files (the PHP file and the XML file), just "zip" them up into a compressed archive file. If your plugin uses a subdirectory, then simply include that in the archive as well.

Loading

Triggering Events

Using the plugin system in your add-on is fairly simple. The most important part is good planning because, to some degree, you're defining an interface for other people to use.

The first thing you need to do is to load your plug-in group. This is done via the following code:

```
JPluginHelper::importPlugin( 'myplugingroup' );
```

This will load all enabled plug-ins that have defined themselves as part of your group. The next thing you need to do is get an instance of the JDispatcher class like so:

```
$dispatcher =& JDispatcher::getInstance();
```

Notice two things here. First, we are using the getInstance() method, not "new" to create a new instance. That is because we need to get the global singleton instance of the JDispatcher object which contains a list of all the plug-ins available. Second, we are using the =& construct to make sure we have a reference to the instance of the JDispatcher and not a copy. Of course this really only applies to PHP version 4, but since you are a good cross-version developer, you will allow for PHP 4 users.

Next, we need to trigger our custom event:

```
$results = $dispatcher->trigger( 'onCdAddedToLibrary', array( &$artist
, &$title ) );
```

Here we have triggered the event 'onCdAddedToLibrary' and passed in the artist name and title of the track. All plug-ins will receive these parameters, process them and optionally pass back information. You can then handle that information however you like.

In summary, here's the complete example code:

```
JPluginHelper::importPlugin( 'myplugingroup' );
$dispatcher =& JDispatcher::getInstance();
```

```
$results = $dispatcher->trigger( 'onCdAddedToLibrary', array( &$artist
, &$title ) );
```

Note: One thing to notice about the trigger method is that there is nothing defining which group of plug-ins should be notified. In actuality, all plug-ins that have been loaded are notified regardless of the group they are in. So, it's important to make sure you have an event name that does not conflict with any other plug-in group's event name. Most of the time this is not an issue because your component is the one that is loading the plug-in group, so you know which ones are loaded, however be aware that the "system" plugin group is loaded very close to the beginning of the request, so you have to make sure you don't have any event naming conflicts with the system events.

Templates

Overview

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.

This article guides you through the process of designing your own template for a HUB. This is intended for web designers/developers with a solid knowledge of CSS and HTML and some basic sense of aesthetics.

Although many currently available HUBs tend to look somewhat similar, you have the freedom to make your HUB look as unique as you want it to be simply by modifying a few CSS and HTML files within your template folder.

Note: All the following articles will refer to construction of a front-end template. However, the concepts, techniques, and methods used also apply to the creation of administrative (back-end) templates unless otherwise noted.

Examples

We have provided an example PhotoShop design file and finished template that you may use to follow along with the articles or use as a starter for your own HUB template.

Download PhotoShop design file (zip)
Download Basic Template (zip)

Installation

Installing

See Installing Extensions for details.

Uninstalling

See Uninstalling Extensions for details.

Designing

Overview

Although many currently available HUBs tend to look somewhat similar, you have the freedom to make your HUB look as unique as you want it to be simply by modifying a few CSS and HTML files within your template folder.

This article makes references to Adobe Photoshop for creation of design files and images but the developer may use any imaging software they're comfortable with.

Creating A Mock-up

It is recommended to start the design of your HUB template by taking a look at a number of other HUBs and websites and deciding which features are important and best serve the goals of your HUB. Having PIs and other team members involved in the process from the start usually saves much time for defining and polishing the design concept. Once you have a good idea of the look and feel of your HUB and its main features, you would normally create a sketch of the HUB front page in Sketch, Figma, Adobe XD, Adobe Photoshop or a similar graphics program. Any secondary page will usually keep the header with the menu and login area, and the footer. Make sure to get feedback from others and finalize the mock-up before jumping onto the next step.

0.8.0

Manifests

Overview

All templates should include a manifest in the form of an XML document named templateDetails.xml. The file holds key "metadata" about the template and is essential. Without it, your template won't be seen by Joomla!.

Directory & Files

Manifests are stored in the same directory as the template file itself and must be named templateDetails.xml.

```
/hubzero
  /templates
    /{TemplateName}
      /css
      /html
      /images
      /js
      error.php
      index.php
      templateDetails.xml
      template_thumbnail.png
      favicon.ico
```

Structure

This XML file just lines out basic information about the template such as the owner, version, etc. for identification by the Joomla! installer and then provides optional parameters which may be set in the Template Manager and accessed from within the module's logic to fine tune its behavior. Additionally, this file tells the installer which files should be copied and installed.

A typical template manifest:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE install PUBLIC "-//Joomla! 1.5//DTD template 1.0//EN"
 "http://dev.joomla.org/xml/1.5/template-install.dtd">
<install version="1.5" type="template">
 <name>mynewtemplate</name>
 <creationDate>2008-05-01</creationDate>
 <author>John Doe</author>
```

```
 <authorEmail>john@example.com</authorEmail>
 <authorUrl>http://www.example.com</authorUrl>
 <copyright>John Doe 2008</copyright>
 <license>GNU/GPL</license>
 <version>1.0.2</version>
 <description>My New Template</description>
 <files>
  <filename>index.php</filename>
  <filename>component.php</filename>
  <filename>templateDetails.xml</filename>
  <filename>template_thumbnail.png</filename>
  <filename>images/background.png</filename>
  <filename>css/style.css</filename>
 </files>
 <positions>
  <position>breadcrumb</position>
  <position>left</position>
  <position>right</position>
  <position>top</position>
  <position>user1</position>
  <position>user2</position>
  <position>user3</position>
  <position>user4</position>
  <position>footer</position>
 </positions>
</install>
```

Let's go through some of the most important tags:

INSTALL
: The install tag has several key attributes. The type must be "template".

NAME
: You can name the templates in any way you wish.

FILES
: The files tag includes all of the files that will will be installed with the template.

POSITIONS
: The module positions used in the template.

The one noticeable difference between this template manifest and the typical manifest of a module or component is the lack of params. While templates may have their own params for further configuration via the administrative back-end, they aren't as commonly found as in other extension manifests. Most HUBzero templates do not include them.

See Joomla!'s Documentation on the full list of available parameter types and what they do.

Page Layout

Overview

A template will typically have two layout files: index.php for the majority of content and error.php for custom error pages ("404 - Not Found", etc.). Both of these files are contained within the top level of a template (i.e., they cannot be placed in a sub-directory of the template).

```
/hubzero
  /templates
    /{TemplateName}
      error.php
      index.php
```

All the HTML that defines the layout of your template is contained in a file named index.php. The index.php file becomes the core of every page that is delivered and, because of this, the file is **required**. Essentially, you make a page (like any HTML page) but place PHP code where the content of your site should go.

The error.php layout, unlike index.php is optional. When not included in a template, Joomla! will use its default system error layout to display site errors such as "404 - Page Not Found". Including error.php is recommended though as it helps give your site a more cohesive feel and experience to the user.

A Breakdown of index.php

Note: For the sake of simplicity, we've excluded some more common portions found in HUBzero templates. The portions removed were purely optional and not necessary for a template to function correctly. We suggest inspecting other templates that may be installed on your HUB for further details.

Starting at the top:

```
<?php
defined( '_JEXEC' ) or die( 'Restricted access' );

// Get the site config
$jconfig =& JFactory::getConfig();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
 xml:lang="<?php echo $this->language; ?>" lang="<?php echo $this->lan
guage; ?>" >
```

The first line prevents unauthorized people from looking at your coding and potentially causing trouble. Then we grab a reference to the global site configuration. The first line of actual HTML tells the browser (and webbots) what sort of page it is. The next line says what language the site is in.

```
<head>
 <!-- This includes metadata tags and the <title> tag -->
 <jdoc:include type="head" />

 <!-- Include the template's main CSS file -->
 <link rel="stylesheet" type="text/css" href="<?php echo $this->baseur
l ?>/templates/<?php echo $this->template; ?>/css/print.css" />
 <!--[if lte IE 7]>
  <link rel="stylesheet" type="text/css" media="screen" href="<?php ec
ho $this->baseurl ?>/templates/<?php echo $this->template; ?>/css/ie7w
in.css" />
 <![endif]-->
 <!--[if lte IE 6]>
  <link rel="stylesheet" type="text/css" media="screen" href="<?php ec
ho $this->baseurl ?>/templates/<?php echo $this->template; ?>/css/ie6w
in.css" />
 <![endif]-->
</head>
```

The first line gets Joomla! to put the correct header information in. This includes the page title, meta information, your main.css, system JavaScript, as well as any CSS or JavaScript that was pushed to the template from an extension (component, module, or plugin). This is a bit different than Joomla! 1.5's typical behavior in that the HUBzero code is automatically finding and including main.css and some key JavaScript files from your template. This is done due to the fact that order of inclusion is important for both CSS and JavaScript. For instance, one cannot execute JavaScript code built using the MooTools framework *before* the framework has been included. It would simply fail. As such, the naming and existence of specific directories, CSS, and JavaScript files becomes quite important for a HUBzero template.

The rest creates links to a print style sheet (if it exist, is named print.css and is located in the

/css folder) and a couple CSS fix style sheets for Internet Explorer (more on this in the Cascading Style Sheets chapter).

Now for the main body:

```
<body>

 <div id="header">
  <h1><a href="<?php echo $this->baseurl ?>" title="<?php echo $jconfi
g->getValue('config.sitename'); ?>"><?php echo $jconfig->getValue('con
fig.sitename'); ?></a></h1>

  <ul id="toolbar" class="<?php if (!$juser->get('guest')) { echo 'log
gedin'; } else { echo 'loggedout'; } ?>">
<?php
 // Get the current user object
 $juser =& JFactory::getUser();

 // Is the user logged in?
 if (!$juser->get('guest')) {
  // Yes. Show them a different toolbar.
  echo '<li id="logout"><a href="/logout"><span>'.JText::_('Logout').'
</span></a></li>';
  echo '<li id="myaccount"><a href="/members/'.$juser->get('id').'"><s
pan>'.JText::_('My Account').'</span></a></li>';
  echo '<li id="usersname">'.$juser->get('name').' ('.$juser->get('use
rname').')</li>';
 } else {
  // No. Show them the login and register options.
  echo "ttt".'<li id="login"><a href="/login" title="'.JText::_('Login
').'">'.JText::_('Login').'</a></li>'."n";
  echo "ttt".'<li id="register"><a href="/register" title="'.JText::_(
'Sign up for a free account').'">'.JText::_('Register').'</a></li>'."n
";
 }
?>
  </ul>

  <!-- Include any modules for the "search" position -->
  <jdoc:include type="modules" name="search" />
 </div><!-- / #header -->

 <!-- Include any modules assigned to the "user3" position -->
 <div id="nav">
  <h2>Navigation</h2>
  <jdoc:include type="modules" name="user3" />
```

```
 </div><!-- / #nav -->

 <div id="wrap">
  <div id="content" class="<?php echo $option; ?>">
   <!-- Include the component output -->
   <jdoc:include type="component" />
  </div><!-- / #content -->

  <div id="footer">
   <!-- Include any modules assigned to the "footer" position -->
   <jdoc:include type="modules" name="footer" />
  </div><!-- / #footer -->
 </div><!-- / #wrap -->
</body>
```

First we layout the site's masthead in the block. Inside, we set the <h1> tag to the site's name, taken from the global site configuration.

Next, we move on to a toolbar that is present in the masthead of every page. This toolbar contains "login" and "register" links when not logged in and "logout" and "My Account" links when logged in. While not required, it is highly recommended that all templates include some form of this arrangement in an easy-to-find, consistent location.

Some modules that have been assigned the position "search" are then loaded in the masthead. Most HUBzero templates default to having a simple search form module appear. Again, this is not required and placement of modules is entirely up to the developer(s) but we, once again, strongly recommend that some form of a search box be included on all pages.

Then we move on to a block where navigation is loaded. It is here that our main menu will appear.

Next, we get to the primary content block. One of the first things you may notice is the use of module as a jdoc:include type. This is how we tell where in our template to output modules that have been assigned to specific positions.

It is also worth noting the small bit of PHP (<?php echo $option; ?>) in the class attribute of the content . This small bit of code outputs the name of the current component as a CSS class. So, if one were on a page of a "groups" component, the resulting HTML would be . Since all component output is contained inside the "content" div, this allows for more specific CSS targeting.

See the Modules: Loading article for more details on module positioning.

The content div contains a very important jdoc:include of type component. This is where all component output will be injected in the template. It is essential this line be included in a template for it to be able to display any content.

Now for the final portion:

```
</html>
<?php
 // Get the page title
 $title = $this->getTitle();
 // Prepend the page title with the site name
 $this->setTitle( $jconfig->getValue('config.sitename').' - '.$title )
;
?>
```

The PHP in the example above is purely optional. What is does is take the current page title and prepends the site's name. Thus, every page results with a title like "myHUB.org - My Page Title".

A Breakdown of error.php

Starting at the top:

```
<?php
defined( '_JEXEC' ) or die( 'Restricted access' );

// Get the site config
$jconfig =& JFactory::getConfig();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
 xml:lang="<?php echo $this->language; ?>" lang="<?php echo $this->lan
guage; ?>" >
```

The first line prevents unauthorized people from looking at your coding and potentially causing trouble. Then we grab a reference to the global site configuration. The first line of actual HTML

tells the browser (and webbots) what sort of page it is. The next line says what language the site is in.

```
<head>
 <meta http-equiv="content-type" content="text/html; charset=utf-8" />
 <title><?php echo $jconfig->getValue('config.sitename'); ?> - <?php e
cho $this->title; ?> - <?php echo $this->error->message ?></title>
 <link rel="stylesheet" type="text/css" media="all" href="<?php echo $
this->baseurl ?>/templates/<?php echo $this->template; ?>/css/error.cs
s" />
</head>
```

Unlike with index.php, we do not include the <jdoc:include type="head" /> tag. Instead, we simply set a single metadata tag to declare the character set and then set the title tag. Next, we include the error.css style sheet, which contains styling just for this layout.

Now for the main body:

```
<body>
 <div id="wrap">
  <div id="header">
   <h1><a href="<?php echo $this->baseurl ?>" title="<?php echo $confi
g->getValue('config.sitename'); ?>"><?php echo $config->getValue('conf
ig.sitename'); ?></a></h1>
  </div>
  <div id="outline">
   <div id="errorbox" class="code-<?php echo $this->error->code ?>">
    <h2><?php echo $this->error->code ?> - <?php echo $this->error->me
ssage ?></h2>

    <p><?php echo JText::_('You may not be able to visit this page bec
ause of:'); ?></p>

    <ol>
     <li><?php echo JText::_('An out-of-
date bookmark/favourite'); ?></li>
     <li><?php echo JText::_('A search engine that has an out-of-
date listing for this site'); ?></li>
     <li><?php echo JText::_('A mis-typed address'); ?></li>
     <li><?php echo JText::_('You have no access to this page'); ?></l
i>
     <li><?php echo JText::_('The requested resource was not found');
?></li>
```

```
     <li><?php echo JText::_('An error has occurred while processing y
our request.'); ?></li>
    </ol>

    <p><?php echo JText::_('If difficulties persist, please contact th
e system administrator of this site.'); ?></p>
   </div><!-- / #errorbox -->

   <form method="get" action="/search">
    <fieldset>
     <?php echo JText::_('Please try the'); ?> <a href="index.php" tit
le="<?php echo JText::_('Go to the home page'); ?>"><?php echo JText::
_('Home Page'); ?></a> <span><?php echo JText::_('or'); ?></span>
     <label>
      <?php echo JText::_('Search:'); ?>
      <input type="text" name="searchword" value="" />
     </label>
     <input type="submit" value="<?php echo JText::_('Go'); ?>" />
    </fieldset>
   </form>
  </div><!-- / #outline -->
<?php
  if ($this->debug) :
   echo "tt".'<div id="techinfo">'."n";
   echo $this->renderBacktrace()."n";
   echo "tt".'</div>'."n";
  endif;
?>
 </div><!-- / #wrap -->
</body>
```

As can be seen, this is relatively straight-forward. We set a title for the page, output the error message, provide some potential reasons for the error and, finally, include a search form. Note that we did not use any modules.

One portion to pay special attention to is the small bit of PHP at the end of the page. This outputs a stack trace when site debugging is turned on.

Note: It is never recommended to turn on debugging on a production site.

Loading Modules

Modules may be loaded in a template by including a Joomla! specific jdoc:include tag. This tag includes two attributes: type, which must be specified as module in this case and name, which specifies the position that you wish to load. Any modules assigned to the specified position (set via the administrative Module Manager) declared in the name attribute will have their output placed in the template (the jdoc:include is removed by Joomla! afterwards).

```
<jdoc:include type="modules" name="footer" />
```

See the Modules: Loading article for further details on how to use more advanced features.

Cascading Style Sheets

Overview

CSS stands for Cascading Style Sheet. HTML tags specify the graphical flow of the elements, be it text, images or flash animations, on a webpage. CSS allows us to define the appearances of those HTML tags with their content, somewhere, so that other pages, if want be, may adhere to. This brings along consistency throughout a website. The cascading effect stipulates that the style of a tag (parent) may be inherited by other tags (children) inside it.

Professional websites separate styling from content. There are many reasons for this, the most obvious (to a developer) being the ability to control the appearance of many pages by changing one file. Styling information includes: fonts, backgrounds, images (that recur on every page), position and dimensions of elements on the page. Your HTML file will now be left with: header information; a series of elements; the text of your website. Because you are creating a Joomla! template, you will actually have: some header information, PHP code to request the rest of the header information, a series of elements, PHP code to request each module position, and PHP code to request the main content.

Style information is coded in CSS and usually stored in files with the suffix .css. A webpage contains a link to the associated .css file so a browser can find the appropriate style information to apply to the page. CSS can also be placed inside a HTML file between <style type="text/css"></style> tags. This is, however, discouraged as it is mixing style and content elements which can make future changes more difficult.

Implementation

Definitions for this section:

External CSS files
: using <link> in the <head>

Document head CSS
: using <style> in the <head>

Inline CSS
: using the style attribute on a tag, i.e.

Guidelines

1. External CSS files should be used in preference to document head CSS and document head CSS should be used in preference to inline CSS.
2. CSS files MUST have the file extension .css and should be stored in the relevant includes directory in the site structure, usually /style/.
3. The file size of CSS files should be kept as low as possible, especially on high demand

pages.
4. External CSS must be linked to using the <link> element which must be placed in the head section of the document. This is the preferred method of using CSS. It offers the best experience for the user as it helps prevent FOUC (flash of unstyled content), promotes code reuse across a site and is cacheable.
5. External style sheets should not be imported (i.e. using @import) as it impairs caching. In IE @import behaves the same as using <link> at the bottom of the page (preventing progressive rendering), so it's best not to use it. Mixing <link> and @import has a negative effect on browsers' ability to asynchronously download the files.
6. Document head CSS may be used where a style rule is only required for a specific page.
7. Inline styles should not be used.
8. Query string data (e.g. "style.css?v=0.1") should not be used on an external CSS file. Use of query strings on CSS files prevents them from caching in some browsers. Whilst this may be desirable for testing, and of course may be used for that, it is very undesirable for production sites.

Directory & Files

Convention places CSS files within a directory named css inside the template directory. While developers are not restricted to this convention, we do recommend it as it helps keep the layout and structure of HUBzero templates consistent. A developer from one project will instantly know where to find certain files and be familiar with the directory structure when working on a project originally developed by someone else.

There are a handful of common CSS files found among most HUBzero. While none of these are required, it is encouraged to follow the convention of including them as it promotes consistency among HUBzero templates and comes with the advantage that certain files, such as main.css are auto-loaded, thus reducing some work on the developer's part.

Here's the standard directory and files for CSS found in a HUBzero template:

```
/hubzero
  /templates
    /{TemplateName}
      /css
        error.css
        ie6.css
        ie7.css
        ie8.css
        main.css
        print.css
        reset.css
```

File details:

error.css
: This is the primary stylesheet loaded by error.php.

ie8.css
: Style fixes for Internet Explorer 8.

ie7.css
: Style fixes for Internet Explorer 7.

ie6.css
: Style fixes for Internet Explorer 6.

main.css
: This is the primary stylesheet loaded by index.php. The majority of your styles will be in here.

print.css
: Styles used when printing a page.

reset.css
: This file is meant to be included **before** any other CSS file. Its purpose is to reduce browser inconsistencies in things like default line heights, margins and font sizes of headings, and so on.

reset.css

This file is meant to be included **before** any other CSS file. Its purpose is to reduce browser inconsistencies in things like default line heights, margins and font sizes of headings, and so on.

The reset styles given here are intentionally very generic. There isn't any default color or background set for the <body> element, for example. Colors and any other styling should be addressed in the template's primary stylesheet after loading reset.css.

```
body,div,dl,dt,dd,ul,ol,li,h1,h2,h3,h4,h5,h6,pre,form,fieldset,input,p
,blockquote,th,td {
 margin:0;
 padding:0;
}
table {
 border-collapse:collapse;
 border-spacing:0;
}
fieldset,img {
 border:0;
}
address,caption,cite,code,dfn,em,strong,th,var {
 font-style:normal;
```

```
 font-weight:normal;
}
ul {
 list-style:none;
}
caption,th {
 text-align:left;
}
h1,h2,h3,h4,h5,h6 {
 font-size:100%;
}
q:before,q:after {
 content:'';
}
```

Typical main.css Structure

main.css controls base styling for your HUB, which is usually further extended by individual component CSS.

We took every effort to organize the main.css in a manner allowing you to easily find a section and a class name to modify. E.g. if you want to change the way headers are displayed, look for "headers" section as indicated by CSS comments. Although you can modify all existing classes, depending on your objectives, it is recommended to avoid modifications to certain sections, as indicated below. While you can add new classes as needed, we caution strongly about removing or renaming any of the existing IDs and classes. Many HUBzero components take advantage of these code styles and any alterations made risk breaking the template display.

Some sections that you are likely to modify:

```
Body - may want to change site background or font family.
Links - pick colors for hyperlinks
Headers - pick colors and font size of headings
Lists - may want to change general list style
Header - you will definitely want to change this
Toolbar - display of username, login/logout links etc.
Navigation - display of main menu
Breadcrumbs - navigation under menu on secondary pages
Extra nav - links that appear on the right-
hand side in multiple components
Footer
```

Sections where you would want to avoid serious modifications:

```
Core classes
Site notices, warnings, errors
Primary Content Columns
Flexible Content Columns
Sub menu - display of tabs in multiple components
```

print.css

This is a style sheet that is used only for printing. It removes unnecessary elements such as menus and search boxes, adjusts any background and font colors as needed to improve readability, and can expose link URLs through generated content (advanced browsers only, e.g. Safari, Firefox).

error.css

This is a style sheet that is used only by the error.php layout. It allows for a more custom styling to error pages such as "404 - Page Not Found".

Internet Explorer

We strongly encourage developers to test their templates in as many browsers and on as many operating systems as possible. Most modern browsers will have little differences in rendering, however, Internet Explorer deserves special mention here.

The most widely used browser, Internet Explorer, is also one of the most lacking in terms of CSS support. Internet Explorer has also, traditionally, handled rendering of block elements, element positioning, and other common tasks a bit differently than many other browsers. As can be expected, this has led to much controversy and discussion on how best to handle such differences. We strongly recommend designing for and testing your templates in alternate browsers such as Safari, Firefox, Chrome, or Opera and then applying fixes to Internet Explorer afterwards. We recommend the use of conditional comments to apply special Internet Explorer only stylesheets.

..1a Conditional Comments

Conditional comments only work in Internet Explorer on Windows, and are thus excellently suited to give special instructions meant only for Internet Explorer on Windows. They are supported from Internet Explorer 5 onwards, and it is even possible to distinguish between versions of the browser.

Conditional comments work as follows:

```
<!--[if IE 6]>
 Special instructions for IE 6 here
<![endif]-->
```

Their basic structure is the same as an HTML comment (<!-- -->). Therefore all other browsers will see them as normal comments and will ignore them entirely. Internet Explorer, however, recognizes the special syntax and parses the content of the conditional comment as if it were normal page content. As such, they can contain any web content you wish to display only to Internet Explorer. While we're using this feature to load CSS files, it can also be used to load JavaScript or display Internet Explorer specific HTML.

Note: Since conditional comments use the HTML comment structure, they can only be included in HTML, and not in CSS files.

Conditional comments support some variation in syntax. For example, it is possible to target a specific browser version as demonstrated above or target multiple versions such as "all versions of Internet Explorer lower than 7". This can be done with a couple handy operators:

- gt = greater than
- gte = greater than or equal to
- lt = less than
- lte = less than or equal to

```
<!--[if IE]>
 According to the conditional comment this is Internet Explorer
<![endif]-->
<!--[if IE 5]>
 According to the conditional comment this is Internet Explorer 5
<![endif]-->
<!--[if IE 5.0]>
 According to the conditional comment this is Internet Explorer 5.0
<![endif]-->
<!--[if IE 5.5]>
 According to the conditional comment this is Internet Explorer 5.5
```

```
<![endif]-->
<!--[if IE 6]>
 According to the conditional comment this is Internet Explorer 6
<![endif]-->
<!--[if IE 7]>
 According to the conditional comment this is Internet Explorer 7
<![endif]-->
<!--[if IE 8]>
 According to the conditional comment this is Internet Explorer 8
<![endif]-->
<!--[if gte IE 5]>
 According to the conditional comment this is Internet Explorer 5 and
up
<![endif]-->
<!--[if lt IE 6]>
 According to the conditional comment this is Internet Explorer lower
than 6
<![endif]-->
<!--[if lte IE 5.5]>
 According to the conditional comment this is Internet Explorer lower
or equal to 5.5
<![endif]-->
<!--[if gt IE 6]>
 According to the conditional comment this is Internet Explorer greate
r than 6
<![endif]-->
```

So, to load stylesheets to specific versions of Internet Explorer in our template we do something like the following:

```
<html>
  <head>
    ... other CSS files ...
    <!--[if IE 7]>
      <link rel="stylesheet" type="text/css" media="screen" href="{Tem
platePath}/{TemplateName}/css/ie7.css" />
    <![endif]-->
    <!--[if lte IE 6]>
      <link rel="stylesheet" type="text/css" media="screen" href="{Tem
platePath}/{TemplateName}/css/ie6.css" />
    <![endif]-->
  </head>
  ...
```

```
</html>
```

Note: Conditional comments used CSS for should be placed inside the <head> tag of a template *after* all other CSS have been linked for their affects to properly take place.

Loading From An Extension

Components

Often a component will have a style sheet of its own. Pushing CSS to the template from a component is quite easy and involves only two lines of code.

```
ximport('xdocument');
XDocument::addComponentStylesheet('com_example');
```

First, we load the Hubzero_Document class. Next we call the static method addComponentStylesheet, passing it the name of the component as the first (and only) argument. This will first check for the presence of the style sheet in the active template's overrides. If found, the path to the overridden style sheet will be added to the array of style sheets the template needs to include in the <head>. If no override is found, the code then checks for the existence of the CSS in the component's directory. Once again, if found, it gets pushed to the template.

Modules

Loading CSS from a module works virtually the same as loading from a component save one minor difference in code. Instead of calling the addComponentStylesheet method, we call the addModuleStylesheet method and pass it the name of the module.

```
ximport('xdocument');
XDocument::addModuleStylesheet('mod_example');
```

Plugins

Loading CSS from a plugin works similarly to loading from a component or module but instead

we call the addPluginStylesheet method and pass it the name of the plugin group **and** the name of the plugin.

```
ximport('xdocument');
XDocument::addPluginStylesheet('examples', 'test');
```

Plugin CSS must be named the same as the plugin and located within a directory of the same name as the plugin inside the plugin group directory.

```
/plugins
  /examples
    /test
      test.css
    test.php
    test.xml
```

Further Help

Resources for learning and sharpening CSS skills:

- CSS Zen Garden
- CSS From The Ground Up
- Guide to Cascading StyleSheets
- CSS School

JavaScript

Overview

Joomla! 1.5 comes with the MooTools 1.11 Javascript Framework. MooTools is not only a visual effects library–it also support Ajax request and JSON notation, table sort, drag & drop operations and much more. All current HUBzero JavaScripts are built on this framework.

Directory & Files

The MooTools framework can be found within the /media/system/js directory. Joomla! includes both a compressed version used for production and an uncompressed version used for debug mode and developer reference.

```
/hubzero
  /media
    /system
      /js
        mootools-uncompressed.js
        mootools.js
```

Most HUBzero templates will include some scripts of their own for basic setup, visual effects, etc. These are generally stored in (but not limited to) a sub-directory, named /js, of the template's main directory.

```
/hubzero
  /templates
    /{TemplateName}
      /js
        globals.js
        hub.js
        modal.js
        tooltips.js
        ...
```

Of the scripts commonly found in a HUBzero template, hub.js and globals.js are perhaps the most important and it is strongly encouraged that developers include these files in their template.

hub.js

If a template includes hub.js, it will be auto-loaded by the system (thus, no reason to specifically declare it in your layout file). When loaded successfully, it will check for the inclusion of the MooTools framework and its version. Should everything pass, the script will then load any other scripts you declare in a comma-separated string.

```
var HUBzero = {
  Version: '1.1',
  require: function(libraryName) {
    // inserting via DOM fails in Safari 2.0, so brute force approach
    document.write('<script type="text/javascript" src="'+libraryName+
'"></script>');
  },
  load: function() {
    if((typeof MooTools=='undefined') ||
      parseFloat(MooTools.version)
      throw("This HUB requires the MooTools JavaScript framework >= 1.
1.0");

    $A(document.getElementsByTagName("script")).each( function(s) {
      if (s.src && s.src.match(/hub.js(?.*)?$/)) {
   var path = s.src.replace(/hub.js(?.*)?$/,'');
      var includes = s.src.match(/?.*load=([a-z,]*)/);
      (includes ? includes[1] : 'globals,tooltips').split(',').each(
       function(include) { HUBzero.require(path+include+'.js') });
   }
    });
  }
}

HUBzero.load();
```

globals.js and the HUB Namespace

Most HUBzero templates will include a global.js file that first establishes a HUB namespace and then proceeds through some basic setup routines. All HUBzero built components, modules, and templates that employ JavaScript place scripts within this HUB namespace. This helps prevent any naming collisions with third-party libraries. While it is recommended that any scripts you may add to your code is also placed within the HUB namespace, it is not required.

Some additional sub-spaces for further organization are available within the HUB namespace.

Separate spaces for Modules, Components, and Plugins are created. Once again, this further helps avoid possible naming/script collisions. Additionally, one more Base space is created for basic setup and utilities that may be used in other scripts.

```
//  Create our namespace
if (!HUB) {
 var HUB = {};

 // Establish a space for setup/init and utilities
 HUB.Base = {};

 // Establish sub-spaces for the various extensions
 HUB.Components = {};
 HUB.Modules = {};
 HUB.Plugins = {};
}
```

To demonstrate adding code to the namespace, below is code from a script in a component named com_example.

```
//  Create our namespace
if (!HUB) {
 var HUB = {};

 // sub-space for components
 HUB.Components = {};
}

// The Example namespace and init method
HUB.Components.Example = {
 init: function() {
  // do something
 }
}

// Initialize the code
window.addEvent('domready', HUB.Components.Example.init);
```

Loading From An Extension

Components

Occasionally a component will have scripts of its own. Pushing JavaScript to the template from a component is quite easy and involves only a few lines of code.

```
// Get the document object
$document =& JFactory::getDocument();
// Check if the file actually exist
if (is_file(JPATH_ROOT.DS.'components'.DS.'com_example'.DS.'example.js
')) {
 // Add the file to the list of scripts to be outputted in the templat
e
 $document->addScript('components'.DS.'com_example'.DS.'example.js');
}
```

First, we load the document object. Next we check for the existence of the JavaScript file we wish to load. If found, we add it to the array of scripts that will be outputted in the <head> of the site template.

Modules

Loading JavaScript from a module is the same as loading from a component save one minor difference: the path to the JavaScript file is obviously different.

```
// Get the document object
$document =& JFactory::getDocument();
// Check if the file actually exist
if (is_file(JPATH_ROOT.DS.'modules'.DS.'mod_example'.DS.'mod_example.j
s')) {
 // Add the file to the list of scripts to be outputted in the templat
e
 $document->addScript('modules'.DS.'mod_example'.DS.'mod_example.js');
}
```

Plugins

Loading JavaScript from a plugin is the same as loading from a component or module save one minor difference: the path to the JavaScript file is obviously different.

```
// Get the document object
$document =& JFactory::getDocument();
// Check if the file actually exist
```

```
if (is_file(JPATH_ROOT.DS.'plugins'.DS.'examples'.DS.'test.js')) {
 // Add the file to the list of scripts to be outputted in the templat
e
 $document->addScript('plugins'.DS.'examples'.DS.'test.js');
}
```

Output Overrides

Overview

There are many competing requirements for web designers ranging from accessibility to legislative to personal preferences. Rather than trying to over-parameterise views, or trying to aim for some sort of line of best fit, or worse, sticking its head in the sand, "Joomla!" has added the potential for the designer to take over control of virtually all of the output that is generated.

Except for files that are provided in the "Joomla!" distribution itself, these methods for customization eliminate the need for designers and developers to "hack" core files that could change when the site is updated to a new version. Because they are contained within the template, they can be deployed to the Web site without having to worry about changes being accidentally overwritten when your System Administrator upgrades the site.

While Joomla! only allows for overriding views and some HTML, HUBzero has extended this functionality to allow for overriding CSS as well. This allows for even more individualistic styling of components and modules on HUBs.

Component Overrides

Note: Not all HUBzero modules will have layouts or CSS that can be overridden.

Layouts

Layout overrides only work within the active template and are located under the /html/ directory in the template. For example, the overrides for "corenil" are located under /templates/corenil/html/.

It is important to understand that if you create overrides in one template, they will not be available in other templates. For example, "rhuk_milkyway" has no component layout overrides at all. When you use this template you are seeing the raw output from all components. When you use the "Beez" template, almost every piece of component output is being controlled by the layout overrides in the template. "corenil" is in between having overrides for some components and only some views of those components.

The layout overrides must be placed in particular way. Using "Beez" as an example you will see the following structure:

```
/templates
  /beez
    /html
      /com_content  (this directory matches the component directory na
me)
```

```
        /articles        (this directory matches the view directory na
me)
          default.php (this file matches the layout file name)
          form.php
```

The structure for component overrides is quite simple: /html/com_{ComponentName}/{ViewName}/{LayoutName}.php.

Sub-Layouts

In some views you will see that some of the layouts have a group of files that start with the same name. The category view has an example of this. The blog layout actually has three parts: the main layout file blog.php and two sub-layout files, blog_item.php and blog_links.php. You can see where these sub-layouts are loaded in the blog.php file using the loadTemplate method, for example:

```
echo $this->loadTemplate('item');
// or
echo $this->loadTemplate('links');
```

When loading sub-layouts, the view already knows what layout you are in, so you don't have to provide the prefix (that is, you load just 'item', not 'blog_item').

What is important to note here is that it is possible to override just a sub-layout without copying the whole set of files. For example, if you were happy with the Joomla! default output for the blog layout, but just wanted to customize the item sub-layout, you could just copy:

```
/components/com_content/views/category/tmpl/blog_item.php
```

to:

```
/templates/rhuk_milkyway/html/com_content/category/blog_item.php
```

When Joomla! is parsing the view, it will automatically know to load blog.php from com_content

natively and blog_item.php from your template overrides.

Cascading Style Sheets

Over-ridding CSS is a little more straight-forward over-ridding layouts. Take the com_groups component for example:

```
/components
  /com_groups
    ...
    com_groups.css   (the component CSS file)
```

To override the CSS, we simply copy or create a new CSS file named the same and place it in the template's overrides:

```
/templates
  /corenil
    /html
      /com_groups   (this directory matches the component directory na
me)
        com_groups.css   (this file matches the CSS file name)
```

To push CSS from a component to the template, add the following somewhere in the component:

```
ximport('xdocument');
XDocument::addComponentStylesheet('com_example');
```

Module Overrides

Note: Not all HUBzero modules will have layouts or CSS that can be overridden.

Layouts

Modules, like components, are set up in a particular directory structure.

```
/modules
  /mod_latest_news
```

```
    /tmpl
      default.php   (the layout)
    helper.php   (a helper file containing data logic)
    mod_latest_news.php   (the main module file)
    mod_latest_news.xml   (the installation XML file)
```

Similar to components, under the main module directory (in the example, mod_latest_news) there is a /tmpl/ directory. There is usually only one layout file but depending on who wrote the module, and how it is written, there could be more.

As for components, the layout override for a module must be placed in particular way. Using "corenil" as an example again, you will see the following structure:

```
/templates
  /corenil
    /html
      /mod_latest_news   (this directory matches the module directory
name)
        default.php   (this file matches the layout file name)
```

Take care with overriding module layout because there are a number of different ways that modules can or have been designed so you need to treat each one individually.

Cascading Style Sheets

Over-ridding CSS files works in precisely the same way as over-ridding layouts. Take the mod_reportproblems module for example:

```
/modules
  /mod_reportproblems
    ...
    mod_reportproblems.css   (the module CSS file)
```

To override the CSS, we simply copy or create a new CSS file named the same and place it in the template's overrides:

```
/templates
```

```
  /corenil
    /html
      /mod_reportproblems   (this directory matches the module directo
ry name)
        mod_reportproblems.css   (this file matches the CSS file name)
```

To push CSS from a module to the template, add the following somewhere in the module:

```
ximport('xdocument');
XDocument::addModuleStylesheet('mod_example');
```

Plugin Overrides

Note: Not all HUBzero plugins will have layouts or CSS that can be overridden.

Layouts

Plugins, like components and modules, are set up in a particular directory structure.

```
/plugins
  /groups
    forum.php   (the main plugin file)
    forum.xml   (the installation XML file)
    /forum
      /views
        /browse
          /tmpl
            default.php   (the layout)
            default.xml   (the layout installation XML file)
```

Similar to components, under the views directory of the plugin's self-titled directory (in the example, forum) there are directories for each view name. Within each view directory is a /tmpl/ directory. There is usually only one layout file but depending on who wrote the plugin, and how it is written, there could be more.

As with components and modules, the layout override for a plugin must be placed in a particular way. Using "corenil" as an example again, you will see the following structure:

```
/templates
  /corenil
    /html
      /plg_groups_forum   (this directory follows the naming pattern o
f plg_{group}_{plugin})
        /browse   (this file matches the layout directory name)
          default.php   (this file matches the layout file name)
```

Take care with overriding plugin layout because there are a number of different ways that plugins can or have been designed so you need to treat each one individually.

Cascading Style Sheets

Over-ridding CSS files works in precisely the same way as over-ridding layouts. Take the forum plugin for groups for example:

```
/plugins
  /groups
    /forum
      forum.css   (the plugin CSS file)
```

To override the CSS, we simply copy or create a new CSS file named the same and place it in the template's overrides:

```
/templates
  /corenil
    /html
      /plg_groups_forum   (this directory follows the naming pattern o
f plg_{group}_{plugin})
        forum.css   (this file matches the CSS file name)
```

To push CSS from a module to the template, add the following somewhere in the module:

```
ximport('xdocument');
XDocument::addPluginStylesheet('groups', 'forum');
```

Pagination Links Overrides

This override can control the display of items-per-page and the pagination links that are used with lists of information. Most HUBzero templates will come with a pagination override that outputs what we feel is a good standard for displaying pagination links and controls. However, feel free to alter this as you see fit. The override can be found here:

```
/templates/{TemplateName}/html/pagination.php
```

When the pagination list is required, Joomla! will look for this file in the default templates. If it is found it will be loaded and the display functions it contains will be used. There are four functions that can be used:

pagination_list_footer
: This function is responsible for showing the select list for the number of items to display per page.

pagination_list_render
: This function is responsible for showing the list of page number links as well at the Start, End, Previous and Next links.

pagination_item_active
: This function displays the links to other page numbers other than the "current" page.

pagination_item_inactive
: This function displays the current page number, usually not hyperlinked.

Quick Reference

Using the corenil template as an example, here is a brief summary of the principles that have been discussed.

Note: Not all HUBzero components, plugins, and modules will have layouts that can be overridden.

Component Output

To override a component layout (for example the default layout in the article view), copy:

```
/components/com_content/views/article/tmpl/default.php
```

to:

```
/templates/corenil/html/com_content/article/default.php
```

To override a component CSS (for example the stylesheet in the com_groups), copy:

```
/components/com_groups/com_groups.css
```

to:

```
/templates/corenil/html/com_groups/com_groups.css
```

To push CSS from a component to the template, add the following somewhere in the component:

```
ximport('Hubzero_Document');
Hubzero_Document::addComponentStylesheet('com_example');
```

Module Output

To override a module layout (for example the Latest News module), copy:

```
/modules/mod_latest_news/tmpl/default.php
```

to:

```
/templates/corenil/html/mod_latest_news/default.php
```

To override a module CSS (for example the stylesheet in the mod_reportproblems), copy:

```
/modules/mod_reportproblems/mod_reportproblems.css
```

to:

```
/templates/corenil/html/mod_reportproblems/mod_reportproblems.css
```

To push CSS from a module to the template, add the following somewhere in the module:

```
ximport('Hubzero_Document');
Hubzero_Document::addModuleStylesheet('mod_example');
```

Plugin Output

To override a plugin layout (for example the Forum plugin for groups), copy:

```
/plugins/groups/forum/views/browse/tmpl/default.php
```

to:

```
/templates/corenil/html/plg_groups_forum/browse/default.php
```

To override a plugin CSS (for example the stylesheet for the forum plugin for groups), copy:

```
/plugins/groups/forum/forum.css
```

to:

```
/templates/corenil/html/plg_groups_forum/forum.css
```

To push CSS from a plugin to the template, add the following somewhere in the plugin:

```
ximport('Hubzero_Document');
Hubzero_Document::addPluginStylesheet('groups', 'forum');
```

Customise the Pagination Links

To customize the way the items-per-page selector and pagination links display, edit the following file:

```
/templates/corenil/html/pagination.php
```

Packaging

Preparation

File Structure

The most basic files, such as index.php, error.php, templateDetails.xml, template_thumbnail.png, favicon.ico should be placed directly in your template folder. The most common is to place images, CSS files, JavaScript files etc in separate folders. Joomla! override files must be placed in folders in the folder "html".

```
/{TemplateName}
  /css
    ... CSS files ...
  /html
    ... Overrides ...
  /images
    ... Image files ...
  /js
    ... JavaScript files ...
  error.php
  index.php
  templateDetails.xml
  template_thumbnail.png
  favicon.ico
```

Thumbnail Preview Image

A thumbnail preview image named template_thumbnail should be included in your template. Image size is 206 pixels in width and 150 pixels high. Recommended file format is PNG.

Packaging

Packaging a template for distribution is easy. Just "zip" up the module directory into a compressed archive file. When the ZIP file is installed, the language file is copied to the appropriate language sub-directory of /language/ and is loaded each time the template is loaded. All of the other files are copied to the /templates/{TemplateName} subfolder of the HUB installation.

Note to Mac OS X users

The Finder's "compress" menu item produces a usable ZIP format package, but with one catch.

It stores the files in AppleDouble format, adding extra files with names beginning with "`._`". Thus it adds a file named "`._templateDetails.xml`, which Joomla 1.5.x can sometimes misinterpret. The symptom is an error message, "XML Parsing Error at 1:1. Error 4: Empty document". The workaround is to compress from the command line, and set a shell environment variable "COPYFILE_DISABLE" to "true" before using "compress" or "tar". See the AppleDouble article for more information.

To set an environment variable on a Mac, open a terminal window and type:

```
export COPYFILE_DISABLE=true
```

Then in the same terminal window, change directories into where your template files reside and issue the zip command. For instance, if your template files have been built in a folder in your personal directory called myTemplate, then you would do the following:

```
cd myTemplate
zip -r myTemplate.zip *
```

Tool Developers

Learn how to create new simulation and modeling tools and publish them on a HUB. Sections include:

- Overview of Tool Development Process
- Using Subversion Source Code Control
- Rappture Toolkit for Creating Graphical User Interfaces
- Rappture web site

Overview

Tool Development Process

Each hub relies on its user community to upload tools and other resources. Hubs are normally configured to allow any user to upload a tool. The process starts with a particular user filling out a web form to register his intent to submit a tool. This tells the hub manager to create a new project area for the tool. The user then uploads code into a Subversion source code repository, and develops the code within a workspace. The user can work alone or with a team of other users. When the tool is ready for testing, the hub manager installs the tool and asks the development team to approve it. Then, the hub manager takes one last look at the tool, and if everything looks good, moves the tool to the "published" state. Of course, a tool can be improved even after it is published, and re-installed, approved, and published over and over again.

The complete process is explained in the tool maintenance documentation for hub managers. Additional details about this process can be found in the following seminars:

- Overview of Tool Development Process
- Using Workspaces
- Using Subversion for Source Code Control

Creating Graphical User Interfaces

If a tool already has a graphical user interface that runs under Linux/X11, then it can be published as-is, usually in a matter of hours. There are two caveats:

- **If the tool relies heavily on graphics, it may not perform very well within HUBzero execution containers.** Our containers run in cluster nodes without graphics cards, and are therefore configured with MESA for software emulation of OpenGL. This has much poorer performance than ordinary desktop computers with a decent graphics card, so frame rates are much lower. Also, all graphics are transmitted to the user's web browser after rendering, again lowering the frame rate. You can expect to achieve a few frames per second in the hub environment--good enough to view and interact with the data, but far below 100 frames/sec that you would normally see on a desktop computer.
- **Tools running within the hub have access to the hub's local file system--not the user's desktop.** Many tools have a *File* menu with an *Open* option. When a user invokes this option within the hub environment, it will bring up a file dialog showing the hub file system. The user won't see his own local files there unless he uploads them first via sftp, webdav, or the hub's importfile command.

The graphical user interface for any tool published in the hub environment can be created using standard toolkits for desktop applications--including Java, Matlab, Python/QT, etc.

If you're looking for an easy way to create a graphical interface for a legacy tool or simple modeling code, check out the Rappture Toolkit that is included as part of HUBzero. Rappture reads a simple XML-based description of a tool and generates a graphical user interface automatically. It interfaces naturally with many programming languages, including C/C++, Fortran, Matlab, Python, Perl, Tcl/Tk, and Ruby. It creates tools that look something like the following:

Rappture was designed for the hub environment and therefore addresses the caveats listed above. All Rappture-based tools have integrated visualization capabilities that take advantage of hardware-accelerated rendering available on the HUBzero rendering farm. Rappture-based tools also include options to upload/download data from the end user's desktop via the importfile/exportfile commands available within HUBzero.

For more details about Rappture, see the following links:

- Rappture Quick Overview
- Developing Scientific Tools for the HUBzero Platform (introductory course with 7 lectures)
- Rappture Reference Manual

Combining Tools

Overview

Some of the tools on any hub are really a collection of 3-5 programs acting like a "workbench" for a particular application. Berkeley Computational Nanoscience Class Tools is one such example. It is really a collection of several separate Rappture-based applications, all running on the same desktop, in the same tool session.

We've created a simple window manager called **nanoWhim** that makes it easy to switch back and forth between several applications on a desktop--without all of the fuss and bother associated with a typical window manager. A tool using nanoWhim looks like this:

The combobox at the top lets users switch between applications. Each window that pops up within an application is managed by a set of tabs.

nanoWhim is based on the Whim window manager written in Tcl/Tk. We needed something like this for nanoHUB to create a very simple tabbed interface, so users could easily switch between a couple of tools within the same tool session. A more comprehensive workflow interface is under development, but this simple solution is sometimes useful.

Flipping between tools

The following example shows a Rappture-based application that popped up a separate Jmol application for molecular visualization. Jmol pops up in its own tab, and you can easily switch back and forth between the original application and the Jmol popup by clicking on the tabs, as shown below:

You can click on the **x** on the Jmol tab to close that application.

You can select another application by using the combobox at the very top of the window. That brings up another Rappture-based application, with a different set of inputs and outputs.

You can run each program independently, and the outputs stay separate. If you flip back to the previous application, it will be sitting just the way you left it.

Configuring nanoWhim

To use nanoWhim, you'll need to create two files in the "middleware" directory for your tool: **nanowhimrc** and **invoke**.

The nanowhimrc File

This file configures the various applications that pop up within the tool session. Here's a very simple example:

```
# set an icon
set.config controls_icon header.gif

# first app is an xterm
start.app "Terminal Window" xterm

# second app is a web browser
start.app "Web Browser" firefox
```

Any line that starts with a pound sign (#) is treated as a comment.

The set.config command configures various aspects of the window manager. Right now, the only useful option is controls_icon, which sets the icon shown in the top-left corner of the window. Note that a relative file name is interpreted with respect to the location of the nanowhimrc file itself. In this case, we've assumed that the image header.gif is sitting in the same directory as nanowhimrc.

The rest of the file contains a series of start.app commands for each application that you want to offer. In this case, the first application is called "Terminal Window" and is just an xterm application. The second application is the Firefox web browser, which we label "Web Browser".

Here's a more realistic example:

```
#
# Customize the nanoWhim window manager
#
set.config controls_icon header.gif
```

```
start.app "Average"
  /apps/rappture/invoke_app -t ucb_compnano -T $dir/../rappture/avg -p
 /apps/java/bin

start.app "Molecular Dynamics (Lennard-Jones)"
  /apps/rappture/invoke_app -t ucb_compnano -T $dir/../rappture/ljmd -
p /apps/java/bin

start.app "Molecular Dynamics (LAMMPS)"
  /apps/rappture/invoke_app -t ucb_compnano -T $dir/../rappture/lammps
 -p /apps/java/bin -p /apps/lammps/lammps-12Feb07/bin

start.app "Monte Carlo (Hard Sphere)"
  /apps/rappture/invoke_app -t ucb_compnano -T $dir/../rappture/hsmc -
p /apps/java/bin

start.app "Ising Simulations"
  /apps/java/bin/java -classpath $dir/../bin MonteCarlo
```

Each start.app command starts a different Rappture-based application. The first argument in quotes is the title of the application, which is displayed in the combobox at the top of the window. The remaining arguments are treated as the Unix command that is invoked to start the application.

The commands shown here all use the /apps/rappture/invoke_app script to invoke a Rappture-based application. The -t argument for that script indicates the project (tool) name. The -T argument indicates which directory contains the Rappture tool.xml file. You can use $dir here to locate the directory relative to the nanowhimrc file. Each -p argument adds a directory onto the execution path (environment variable $PATH), which may be needed for simulators and other tools invoked by the Rappture program.

The invoke File

The nanowhimrc file configures the window manager, but the middleware/invoke script actually invokes it. Every tool on nanoHUB has its own invoke script sitting in the middleware directory. Your invoke script should look like this if you want to use nanoWhim:

```
#!/bin/sh
/apps/nanowhim/invoke_app -t ucb_compnano
```

This script invokes the nanoWhim window manager for the project specified by the -t argument.

This is the short name that you gave when you registered your tool with nanoHUB. This script looks for the middleware/nanowhimrc file within your source code, and launches nanoWhim with that configuration.

Testing Your Tool

Normally, you develop and test tools within a workspace in your hub. If you're using nanoWhim, that's still true for the individual applications. In other words, you can test each application individually within a workspace. But to get the full effect of the nanoWhim manager running all applications at once, you'll have to get your tool to "installed" status, and then launch the application in test mode. For details about doing this, see the tool maintenance documentation for hub managers or the lecture on Uploading and Publishing New Tools. Look at the tool status page for your own tool project and find the *Launch Tool* button. This is what you would normally do to test any tool before approving it. Once you're in the "installed" stage and you're able to click *Launch Tool*, the nanoWhim configuration should take effect and you'll be able to test the overall combined tool.

Accessing the Grid

Overview

Tools are hosted within a "tool session" running within the hub environment. The tool session supports the graphical interface, which helps the user set up the problem and visualize results. If the underlying calculation is fairly light weight (e.g., runs in a few minutes or less), then it can run right within the same tool session. But if the job is more demanding, it can be shipped off to another machine via the "submit" command, leaving the tool session host less taxed and more responsive.

This chapter describes the "submit" command, showing how it can be used at the command line within a workspace and also within Rappture-based tools.

Submit Command

Overview

submit takes a user command and executes it remotely. The objective is to allow the user to issue a command in the same manner as a locally executed command. Multiple submission mechanisms are available for job dissemination. A set of steps are executed for each job submission:

- Destination site is selected
- A wrapper script is generated for remote execution
- If needed a batch system description file is generated.
- Input files for a job are gathered and transferred to the remote site. Transferred files include the wrapper script and batch description scripts.
- The wrapper script is executed remotely.
- Progress of the remote job is monitored until completion.
- Output files from the job are returned to the dissemination point.

Command Syntax

submit command options can be determined by using the help parameter of the submit command.

```
$ submit --help
usage: submit [options]

options:
  -v, --venue           Remote job destination
  -i, --inputfile       Input file
  -n NCPUS, --nCpus=NCPUS
                        Number of processors for MPI execution
  -N PPN, --ppn=PPN     Number of processors/node for MPI execution
  -w WALLTIME, --wallTime=WALLTIME
                        Estimated walltime hh:mm:ss or minutes
  -e, --env             Variable=value
  -m, --manager         Multiprocessor job manager
  -h, --help            Report command usage

Currently available DESTINATIONs are:
   clusterA
   clusterB

Currently available MANAGERs are:
   mpich-intel32
```

By specifying a suitable set of command line parameters it is possible to execute commands on configured remote systems. The simple premise is that a typical command line can be prefaced by submit and its arguments to execute the command remotely.

```
$ submit -v clusterA echo Hello world!
Hello world!
```

In this example the echo command is executed on the venue named clusterA where jobs are executed directly on the host. Execution of the same command on a cluster using PBS would be done in a similar fashion

```
$ submit -v clusterB echo Hello world!
(2586337) Simulation Queued Wed Oct  7 14:45:21 2009
(2586337) Simulation Done Wed Oct  7 14:54:36 2009
$ cat 00577296.stdout
Hello world!
```

submit supports an extensible variety of submission mechanisms. HUBzero supported submission mechanisms are

- local - use job submission mechanisms available directly on the submit host. These include PBS and condor job submission.
- ssh - direct use of ssh with pre-generated key.
- ssh + remote batch job submission - use ssh to do batch job submission remotely, again with pre-generated key.

A site for remote submission occurs is selected in one of the following ways, listed in order of precedence:

- User specified on the command line with -v/--venue option.
- Randomly selected from remote sites associated pre-staged application.

Any files specified by the user plus internally generated scripts are packed into a tarball for delivery to the remote site. Individual files or entire directory trees may be listed as command inputs using the -i/--inputfile option. Additionally command arguments that

exist as files or directories will be packed into the tarball. If using ssh based submission mechanisms the tarball is transferred using scp.

The job wrapper script is executed remotely either directly or as a batch job. The job is subject to all remote queuing restrictions and idiosyncrasies.

Remote batch jobs are monitored for progress. Changes in job status are reported at least every minute. Job status is reported at least every four minutes. The job status is used to detect job completion.

The same methods used to transfer input files are applied in reverse to retrieve output files. Any files and directories created or modified by the application are be retrieved. A tarball is retrieved and expanded to the home base directory. It is up to the user to avoid the overwriting of files.

In addition to the application generated output files additional files are generated in the course of remote job execution. Some of these files are for internal bookkeeping and are consumed by submit, a few files however remain in the home base directory. The remaining files include JOBID.stdout and JOBID.stderr, it is also possible that a second set of standard output/error files will exist containing the output from the batch job submission script. JOBID represents unique job identifier assigned by submit.

Rappture Integration with Submit

Overview

It is possible to use the submit command to execute simulation jobs generated by Rappture interfaces remotely. A common approach is to create a shell script which can exec'd or forked from an application wrapper script. This approach has been applied to TCL, Python, Perl wrapper scripts. To avoid consumption of large quantities of remote resources it is imperative that the submit command be terminated when directed to do so by the application user (Abort button).

TCL Wrapper Script

submit can be called from a TCL Rappture wrapper script for remote batch job submission. An example of what code to insert in the wrapper script is detailed here.

An initial code segment is required to catch the Abort button interrupt. Setting execctl to 1 will terminate the process and any child processes.

```
package require RapptureGUI
Rappture::signal SIGHUP sHUP {
   puts "Caught SIGHUP"
   set execctl 1
}
```

A second code segment is used to build an executable script that can executed using Rappture::exec. The trap statement will catch the interrupt thrown when the wrapper script execution is Aborted. Putting the submit command in the background allows for the possibility of issuing multiple submit commands from the script. The wait statement forces the shell script to wait for all submit commands to terminate before exiting.

```
   set    submitScript "#!/bin/sh\n\n"
   append submitScript "trap cleanup HUP INT QUIT ABRT TERM\n\n"
   append submitScript "cleanup()\n"
   append submitScript "{\n"
   append submitScript "   kill -TERM `jobs -p`\n"
   append submitScript "   exit 1\n"
   append submitScript "}\n\n"

   append submitScript "cd [pwd]\n"
   append submitScript "submit -v cluster -n $nodes -w $walltime\\\n"
```

```
append submitScript "       COMMAND ARGUMENTS &\n"
append submitScript "sleep 5\n"
append submitScript "wait\n"

set submitScriptPath [file join [pwd] submit_script.sh]
set fid [open $submitScriptPath w]
puts $fid $submitScript
close $fid
file attributes $submitScriptPath -permissions 00755
```

The standard method for wrapper script execution of commands can now be used. This will stream the output from all submit commands contained in submit_script.sh to the GUI display. The same output will be retained in the variable out.

```
set status [catch {Rappture::exec $submitScriptPath} out]
```

Each submit command creates files to hold COMMAND standard output and standard error. The file names are of the form JOBID.stdout and JOBID.stderr, where JOBID is an 8 digit number. These results can be gathered as follows.

```
set out2 ""
foreach errfile [glob -nocomplain *.stderr] {
   if [file size $errfile] {
      if {[catch {open $errfile r} fid] == 0} {
         set info [read $fid]
         close $fid
         append out2 $info
      }
   }
   file delete -force $errfile
}
foreach outfile [glob -nocomplain *.stdout] {
   if [file size $outfile] {
      if {[catch {open $outfile r} fid] == 0} {
         set info [read $fid]
         close $fid
         append out2 $info
      }
   }
   file delete -force $outfile
}
```

The script file should be removed.

```
file delete -force $submitScriptPath
```

The output is presented as the job output log.

```
$driver put output.log $out2
```

All other result processing can proceed as normal.

Python Wrapper Script

submit can be called from a python Rappture wrapper script for remote batch job submission. An example of what code to insert in the wrapper script is detailed here.

An initial code segment is required to catch the Abort button interrupt.

```
import os
import stat
import Rappture
import signal

def sig_handler(signalNumber, frame):
   if Rappture.tools.commandPid > 0:
      os.kill(Rappture.tools.commandPid,signal.SIGTERM)

signal.signal(signal.SIGINT, sig_handler)
signal.signal(signal.SIGHUP, sig_handler)
signal.signal(signal.SIGQUIT, sig_handler)
signal.signal(signal.SIGABRT, sig_handler)
signal.signal(signal.SIGTERM, sig_handler)
```

A second code segment is used to build an executable script that can executed using Rappture.tools.getCommandOutput. The trap statement will catch the interrupt thrown when the wrapper script execution is Aborted. Putting the submit command in the background allows for the possibility of issuing multiple submit commands from the script. The wait statement forces the shell script to wait for all submit commands to terminate before exiting.

```
   submitScriptName = 'submit_app.sh'
   submitScript     = """#!/bin/sh

trap cleanup HUP INT QUIT ABRT TERM
```

```
cleanup()
{
   echo "Abnormal termination by signal"
   kill -s TERM `jobs -p`
   exit 1
}

"""
   submitScript += "cd %s\\\n" % (os.getcwd())
   submitScript += "submit -v cluster -n %s -w %s \\\n" % (nodes,wallt
ime)
   submitScript += "       %s %s &\\\n" % (COMMAND,ARGUMENTS)
   submitScript += "wait\\\n"

   submitScriptPath = os.path.join(os.getcwd(),submitScriptName)
   fp = open(submitScriptPath,'w')
   if fp:
      fp.write(submitScript)
      fp.close()

   os.chmod(submitScriptPath,
            stat.S_IRWXU|stat.S_IRGRP|stat.S_IXGRP|stat.S_IROTH|stat.S
_IXOTH)
```

The standard method for wrapper script execution of commands can now be used. This will stream the output from all submit commands contained in submit_script.sh to the GUI display. The same output will be retained in the variable stdOutput.

```
   exitStatus,stdOutput,stdError =
               Rappture.tools.getCommandOutput(submitScriptPath)
```

Each submit command creates files to hold COMMAND standard output and standard error. The file names are of the form JOBID.stdout and JOBID.stderr, where JOBID is an 8 digit number. These results can be gathered as follows.

```
   re_stdout = re.compile(".*\.stdout$")
   re_stderr = re.compile(".*\.stderr$")

   out2 = ""
   errFiles = filter(re_stderr.search,os.listdir(os.getpwd()))
   if errFiles != []:
      for errFile in errFiles:
         errFilePath = os.path.join(os.getpwd(),errFile)
```

```
         if os.path.getsize(errFilePath) > 0:
            f = open(errFilePath,'r')
            outFileLines = f.readlines()
            f.close()
            stderror = ''.join(outFileLines)
            out2 += '\n' + stderror
         os.remove(errFilePath)

   outFiles = filter(re_stdout.search,os.listdir(os.getpwd()))
   if outFiles != []:
      for outFile in outFiles:
         outFilePath = os.path.join(os.getpwd(),outFile)
         if os.path.getsize(outFilePath) > 0:
            f = open(outFilePath,'r')
            outFileLines = f.readlines()
            f.close()
            stdoutput = ''.join(outFileLines)
            out2 += '\n' + stdoutput
         os.remove(outFilePath)
```

The script file should be removed.

```
   os.remove(submitScriptPath)
```

The output is presented as the job output log.

```
   lib.put("output.log", out2, append=1)
```

All other result processing can proceed as normal.

Perl Wrapper

submit can be called from a perl Rappture wrapper script for remote batch job submission. An example of what code to insert in the wrapper script is detailed here.

An initial code segment is required to catch the Abort button interrupt.

```
use Rappture

my $ChildPID = 0;

sub trapSig {
```

```
    print "Signal @_ trapped\n";
    if($ChildPID != 0) {
        kill 'TERM', $ChildPID;
        exit 1;
    }
}
$SIG{TERM} = \&trapSig;
$SIG{HUP}  = \&trapSig;
$SIG{INT}  = \&trapSig;
```

A second code segment is used to build an executable script that can executed using Rappture.tools.getCommandOutput. The trap statement will catch the interrupt thrown when the wrapper script execution is Aborted. The wait statement forces the shell script to wait for the submit command to terminate before exiting.

```
$SCRPT = "submit_app.sh";
open(FID,">$SCRPT");
print FID "#!/bin/sh\n";
print FID "\n";
print FID "trap cleanup HUP INT QUIT ABRT TERM\n\n";
print FID "cleanup()\n";
print FID "{\n";
print FID "   kill -s TERM `jobs -p`\n";
print FID "   exit 1\n";
print FID "}\n\n";

print FID "submit -v cluster -n $nPROCS -w $wallTime COMMAND ARGUMENTS
 &\n";
print FID "wait %1\n";
print FID "exitStatus=\$?\n";
print FID "exit \$exitStatus\n";
close(FID);
chmod 0775, $SCRPT;
```

The standard fork and exec method for wrapper script execution of commands can now be used. Using this approach does not allow streaming of the command outputs.

```
if      (!defined($ChildPID = fork())) {
    die "cannot fork: $!";
} elsif ($ChildPID == 0) {
    exec("./$SCRPT") or die "cannot exec $SCRPT: $!";
    exit(0);
```

```
} else {
    waitpid($ChildPID,0);
}
```

Each submit command creates files to hold COMMAND standard output and standard error. The file names are of the form JOBID.stdout and JOBID.stderr, where JOBID is an 8 digit number. These results can be gathered with standard perl commands for file matching, reading, etc. All other result processing can proceed as normal.

System Administrators

The Setup and Maintenance Guide details hardware and software requirements, how to bring up a new HUB, upgrade system software, etc.

Installation

What is HUBzero?

HUBzero is a platform used to create dynamic web sites for scientific research and educational activities. With HUBzero, you can easily publish your research software and related educational materials on the web. Powerful middleware serves up interactive simulation and modeling tools via your web browser. These tools can connect you with rendering farms and powerful Grid computing resources.

Minimum System Requirements

HUBzero installations require one or more dedicated physical hosts running Debian GNU/Linux 5.0.

Other distributions might theoretically work with some modification, although they would be totally unsupported.

A typical starter HUBzero installation might consist of a single physical server with dual 64-bit quad-core CPUs, 16 Gigabytes of RAM and a terabyte of disk

It is possible to run HUBzero inside of a virtual machine such as ones created by VMware and VirtualBox. While fully functional there will would significant performance and resource limitations in such an environment.

Target Audience

This document and the installation of a HUBzero system has a target audience of experienced Linux administrators (preferably experienced with Debian GNU/Linux).

Debian GNU/Linux

Install Basic Operating System

The latest version of Debian GNU/Linux 5.0 (5.0.5 as of this writing) should be installed on each physical host used by a HUBzero installation.

To install Debian GNU/Linux, you can easily obtain a copy, and then follow the installation instructions for your architecture (only 64bit [AMD64] is currently supported).

Installing Debian GNU/Linux using a a small bootable CD is the simplest method.

When installing Debian GNU/Linux be sure to do the following:

- Ensure the disk(s) are partitioned to have at least:
    - A bootable partition at least 100.0 GB in size for the root filesystem.
    - An empty partition at least 50.0 GB in size (note the device name of this partition for later)
    - An appropriately sized swap partition.
- When prompted to select an installation package just select "Standard System", other packages will be added later

When the installation is complete your system will reboot into a minimal Debian GNU/Linux system.

Don't forget to remove your installation media and/or change your server's boot media order if you changed them prior to installation.

Set hostname

Optional. If you didn't specify the fully qualified domain name when running setup you will need to set it here.

HUBzero expects the `hostname` command to return the fully qualified hostname for the system.

```
# hostname myhub.org
```

To make the change permanent you must also edit the file /etc/hostname, this can simply with:

```
# echo "myhub.org" > /etc/hostname
```

Delete local user

If you created a local user account when installing the operating system now would be a good time to delete it before it causes you future problems.

```
# deluser username
```

Configure Networking

Optional. If you didn't configure networking during installation you will need to do so now.

For help with networking setup try this link.

Setting up your IP address.

The IP addresses associated with any network cards you might have are read from the file **/etc/network/interfaces**. This file has documentation you can read with:

```
# man interfaces
```

A sample entry for a machine with a static address would look something like this:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
 address 192.168.1.90
 gateway 192.168.1.1
 netmask 255.255.255.0
 network 192.168.1.0
 broadcast 192.168.1.255
```

Here we've setup the IP addresss, the default gateway, and the netmask.

For a machine running DHCP the setup would look much simpler:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface - use DHCP to find our address
auto eth0
iface eth0 inet dhcp
```

(If you're using a DHCP based setup you must have a DHCP client package installed - usually one of pump or dhcp-client.)

If you make changes to this file you can cause them to take effect by running:

```
# /etc/init.d/networking restart
```

Setting up DNS

Use whatever nameserver and other options as recommended by your ISP. If you used DHCP to set up networking it is likely this has already been set.

When it comes to DNS setup Debian doesn't differ from other distributions. To cause your machine to consult with a particular server for name lookups you simply add their addresses to /etc/resolv.conf.

For example a machine which should perform lookups from the DNS server at IP address 192.168.1.10 would have a resolv.conf file looking like this:

```
nameserver 192.168.1.10
```

Configure Advanced Package Tool

Now configure what debian distribution mirror to use and the location of the HUBzero package repository by editing /etc/apt/sources.list to look like:

```
deb http://ftp.us.debian.org/debian/ lenny main
deb-src http://ftp.us.debian.org/debian/ lenny main
```

```
deb http://security.debian.org/ lenny/updates main
deb-src http://security.debian.org/ lenny/updates main

deb http://volatile.debian.org/debian-volatile lenny/volatile main
deb-src http://volatile.debian.org/debian-volatile lenny/volatile main

deb http://packages.hubzero.org/deb lenny main contrib non-free
deb-src http://packages.hubzero.org/deb lenny main contrib non-free
```

You will need to get and install the hubzero archive key to be able to verify packages from the hubzero archive:

```
# wget http://packages.hubzero.org/deb/hubzero-signing-
key.asc -q -O - | apt-key add -
```

Once the public key for http://packages.hubzero.org has been install you can then upgrade the current packages to their latest releases.

```
# apt-get update
# apt-get upgrade
```

SSH

Next we install fail2ban and ssh

```
# apt-get install fail2ban ssh
```

At this point you can continue configuration and setup remotely if that is more convenient.

Enable OpenVZ

If you are installing this in a VirtualBox VM you must enable PAE/NX support. Go to system ->

processor of your VM, select "Enable PAE/NX".

To use OpenVZ you must use an OpenVZ enabled kernel which is easily installed.

HUBzero makes extensive use of OpenVZ containers so it is recommended to just use the OpenVZ enabled kernel on all HUBzero servers.

```
# apt-get install linux-image-2.6-openvz-amd64
```

You will need to reboot the server to activate the new kernel.

```
# reboot
```

Once you have rebooted you can verify the new kernel is active

```
# uname -a
Linux myhub.hubzero.org 2.6.26-2-openvz-
amd64 #1 SMP Thu Nov 5 03:06:00 UTC 2009 x86_64 GNU/Linux
```

With the new kernel active you can remove the old one

```
# apt-get purge linux-image-2.6.26-2-amd64
```

Prepare Fileystem

The root filesystem ('/') runs with quotas disabled and contains the primary operating system for the server and for each OpenVZ container hosted on the server.

Each HUBzero server may use an addition partition for use appropriate to the function of the server (web document root, project data, home directories, etc).

If you did not create an empty partition during setup, create one now using your favorite disk partitioning tool. Be sure to note the device name for the partition you create as it will be used below.

Once you have an empty partition ready we can install a filesystem. Replace "/dev/PART" with the device name for the empty partition you have created (e.g., /dev/sda2). The command "fdisk -l" will list all paritions the system knows about.

```
# mke2fs -j /dev/PART
# e2fsck -f -C 0 /dev/PART
```

```
# mkdir /export
```

Then make sure the following line appears in /etc/fstab

```
/dev/PART     /export      ext3    defaults,quota,errors=remount-
ro     0     2
```

Then mount the new filesystem

```
# mount /export
```

Bind mount /home

Create a 'home' directory in our new /export filesystem. move the contents of the default home directory to the new location, then bind mount new location over the old.

```
# mkdir -p /export/home/myhub
# mv /home/* /export/home
# mount --bind /export/home /home
```

Bind mount /opt

Currently HUBzero uses the /opt directory for storing subversion and trac data as well as some of hubzero supporting software as well. We recognize this may not be the best organization.

Create a 'opt' directory in our new /export filesystem. move the contents of the default /opt directory to the new location, then bind mount new location over the old.

```
# mkdir /export/opt
# mv /opt/* /export/opt
# mount --bind /export/opt /opt
```

Bind mount /apps

Currently HUBzero uses the /apps directory for storing installed tools and other software that needs to be available to each execution container.

Create a 'apps' directory in our new /export filesystem and in the root filesystem. Then bind mount /export/apps over /apps.

```
# mkdir /export/apps
# mkdir /apps
# mount --bind /export/apps /apps
```

Bind mount /www

HUBzero uses the /www directory for storing the document root and supporting directories needed by the web server.

Create a 'www' directory in our new /export filesystem and in the root filesystem. Then bind mount /export/www over /www.

```
# mkdir -p /export/www
# mkdir /www
# mount --bind /export/www /www
```

Update /etc/fstab

Now edit **/etc/fstab** with the bind mounts we created above by adding the following lines

```
/export/opt  /opt        none    bind,defaults
0     0
/export/apps /apps       none    bind,defaults
0     0
/export/home /home       none    bind,defaults
0     0
/export/www  /www        none    bind,defaults
0     0
```

OpenLDAP

Install OpenLDAP

Install OpenLDAP

```
# apt-get install slapd
```

You will be prompted for an administrative password. This will be the LDAP administrator password and will be used anywhere that write permission to LDAP is required. This will get set again in the next step when we reconfigure OpenLDAP.

Reconfigure OpenLDAP

Debian's default configuration for OpenLDAP is sometimes not quite what you might want. If you want to use an LDAP base DN based off something other than the domain name used when configuring the host you will need to reconfigure the package.

```
# dpkg-reconfigure slapd
```

The reconfiguration script will allow you to change the LDAP base DN to be based on a different domain name. For example, "myhub.org" would become "dc=myhub,dc=org".

The reconfiguration script will then ask for the organization to use. This isn't important to us and can be set arbitrarily.

You will then be asked to enter a password for the admin account. You will need to remember this password for later configuration steps.

Accept the default "HDB" database backend type.

Do not remove database when slapd is purged. Sometimes during maintenance it can be useful to reinstall slapd without wiping out the database.

Move the old database out of the way.

Don't allow LDAPv2 protocol.

Install nscd

The Name Service Cache (nscd) will used later so we go ahead and install it here.

```
# apt-get install nscd
```

Install HUBzero LDAP Schema

```
# apt-get install hubzero-openldap
```

To enable this new schema edit **/etc/ldap/slapd.conf** and add the following line as the last "include" statue under the "Schema and objectClass definitions" comment toward the beginning of the file.

```
include         /etc/ldap/schema/hub.schema
```

Then restart OpenLDAP

```
# /etc/init.d/slapd restart
```

Initialize OpenLDAP Database

Several entries are expected to be prepopulated in OpenLDAP.

There is a script to do this, but the script has to be manually configured.

```
# cp /usr/lib/hubzero/openldap/HUB-INIT-SLAPD.tmpl HUB-INIT-SLAPD
```

Modify HUB-INIT-SLAPD and edit the five configuration lines near the beginning of the file:

```
base_dn=""
admin_pass=""
hubadmin_passhash=""
hubrepo_passhash=""
home_dir=""
```

- **base_dn** should be the base DN of your LDAP (e.g., "dc=myhub,dc=org")

- **admin_pass** should be the clear text password you set for the LDAP administrator.
- **hubadmin_passhash** should be the hashed password for the about to be created hubadmin account. You can hash a password using '/usr/sbin/slappasswd'
- **hubrepo_passhash** should be the hashed password for the about to be created hubrepo account. You can hash a password using '/usr/sbin/slappasswd'
- **home_dir** should be the home directoy you created earlier (eg., "/home/myhub").

Then run the configuration script

```
# sh ./HUB-INIT-SLAPD
```

This should prepopulate the databae enough to bootstrap HUBzero.

Configure PAM to use LDAP

```
# apt-get install libpam-ldap
```

- Use "ldap://127.0.0.1" as the URI
- Set the base DN to match how you configured OpenLDAP (eg., "dc=myhub,dc=org")
- Use LDAP v3
- Accept making local root data admin
- LDAP does not require login
- Specify the DN for the LDAP admin user (eg., "cn=admin,dc=myhub,dc=org")
- Enter admin password for LDAP

Modify **/etc/pam.d/common-auth** by commenting out the existing configuration then adding rules to allow authentication against LDAP.

```
#auth required pam_unix.so nullok_secure

auth sufficient pam_unix.so nullok_secure
auth sufficient pam_ldap.so try_first_pass
auth required pam_deny.so
```

Modify **/etc/pam_ldap.conf** by adding the following section (other mappings in this file should already be commented out).

```
# HUBzero Mappings
nss_base_passwd ou=users,?one
nss_base_shadow ou=users,?one?host=web
pam_filter host=web
pam_password crypt
```

```
nss_map_attribute uniqueMember member
nss_base_group ou=groups,dc=myhub,dc=org?sub
```

Be sure the BASEDN in the above matches that used by your configuration.

/etc/pam_ldap.secret contains the LDAP admin password and should only be readable by root.

Configure NSS to use OpenLDAP

```
# apt-get install libnss-ldap
```

- Specify the DN for the ldap admin account
- Specify the password for the ldap admin account

Modify **/etc/libnss-ldap.conf** by adding the following section. (other mappings in this file should already be commented out).

```
# HUBzero Mappings
nss_base_passwd ou=users,?one
nss_base_shadow ou=users,?one?host=web
pam_filter host=web
pam_password crypt
nss_map_attribute uniqueMember member
nss_base_group ou=groups,dc=myhub,dc=org?sub
```

Be sure the BASEDN in the above matches that used by your configuration.

Modify **/etc/nsswitch.conf**

```
passwd:         compat ldap
group:          compat ldap
shadow:         compat ldap
```

/etc/libnss-ldap.secret contains the LDAP admin password and should only be readable by root.

Test

```
# getent passwd
```

To test configuration. You should see entries for users 'hubrepo' and 'apps' toward the end of the list if everything is working correctly.

Troubleshooting

If you have a problem with the system apparently not recognizing up to date account or group information (eg., in the next section some people report receiving an error about unknown username 'hubadmin') you can nscd to flush it data cache and restart using the following commands:

```
# nscd -i passwd
# nscd -i group
# /etc/init.d/nscd restart
# getent passwd
```

If you still don't see the hubadmin account listed then re-read the instructions and check your work very carefully. These instructions assume a fresh install, if you are working with an existing LDAP/PAM/NSS installation you will have to do more advanced troubleshooting outside the scope of this documentation.

Create home directories

Create a home directory for the apps user

```
# mkdir /home/myhub/apps
# chown apps.public /home/myhub/apps
# chmod 0700 /home/myhub/apps
```

MySQL

Install MySQL Server

Install MySQL Server

```
# apt-get install hubzero-mysql
```

You will be prompted for an administrative password. This will be the MySQL administrator password. This will also fix the GRANT permissions on the default Debian debian-sys-maint mysql account.

Apache

Install Apache Web Server

Install Apache Web Server

```
# apt-get install hubzero-apache2
```

The default apache web site should now work and display "It Works!"

Enable default SSL site

```
# a2ensite default-ssl
# /etc/init.d/apache2 restart
```

The default apache ssl web site should now work (be sure to use https:// prefix) and display "It Works!"

The SSL certficate used by the default-ssl and (see next sections) the hub-ssl sites use the self signed "snakeoil" certificate that was installed by the ssl-cert package. This should only be used for testing and development. A commercial certificate should be purchased and installed for any site put into production.

Enable basic hub site

Enable the hub site, while disabling the apache default site.

```
# a2dissite default
# a2ensite hub
# /etc/init.d/apache2 reload
```

This configuration continues to use the default apache document root so the site should display the standard default "It Works!" page

It would be a good idea to restrict access to the web server via a firewall now. A web based installer will be installed later and it should only be accessed by the person setting up the site.

Enable basic hub SSL site

Enable the hub-ssl site, while disabling the default-ssl site.

```
# a2dissite default-ssl
# a2ensite hub-ssl
# /etc/init.d/apache2 reload
```

This configuration continues to use the default apache document root so the https site should display the standard default "It Works!" page

It would be a good idea to restrict access to the web server via a firewall now. A web based installer will be installed later and it should only be accessed by the person setting up the site.

PHP

Configure PHP

Edit **/etc/php5/apache2/php.ini** and set the display_errors parameter to 'Off' and log_errors to "On".

```
display_errors = Off
log_errors = On
```

then restart apache to enable everything.

```
# /etc/init.d/apache2 restart
```

Test

```
# echo "<?php phpinfo();?>" > /var/www/index.php
```

Go to "/index.php" on your site and you should see a php status page.

Delete the test page when you are done.

```
# rm /var/www/index.php
```

Mail

Install

We need to reconfigure exim4 to enable outgoing email (exim4 got installed earlier as a prerequisite for the mysql server).

```
# dpkg-reconfigure exim4-config
```

- Select "internet site; mail is sent and received directly using SMTP" then configure as appropriate for your site.
- Enter the FQDN of your site when asked
- Listen on all IP addresses (i.e., make list blank)
- Other destinations for which mail is accepted: should be made blank
- Domains to relay mail for: should be made blank
- Machines to relay mail for: should be made blank
- Keep num,ber of DNS-queries minimal (Dial-on-Demand): No
- Delivery method for local mail: mbox format in /var/mail/
- Split configuration into small files? No

This is just an example. Mail should be configured however the site needs. The CMS just expects to be able send outgoing email.

CMS

Global HUBzero Configuration

Edit/create the file **/etc/hubzero.conf**

```
[default]
site=myhub

[myhub]
DocumentRoot=/www/myhub
```

Install

Install the content management system.

```
# apt-get install hubzero-cms
# mkdir /www/myhub
# cp -rp /usr/lib/hubzero/cms/* /www/myhub
# chown -R www-data.www-data /www/myhub
```

Note: /www/myhub will be the document root of your site

Create Database

Create database for the HUBzero CMS to use

```
# /usr/bin/mysql --defaults-file=/etc/mysql/debian.cnf
Welcome to the MySQL monitor.  Commands end with ; or g.
Your MySQL connection id is 33
Server version: 5.0.51a-24+lenny2 (Debian)

Type 'help;' or 'h' for help. Type 'c' to clear the buffer.

mysql> CREATE DATABASE `myhub`;
Query OK, 1 row affected (0.00 sec)

mysql> CREATE DATABASE `myhub_metrics`;
Query OK, 1 row affected (0.00 sec)
```

```
mysql> GRANT ALL PRIVILEGES ON `myhub`.* TO 'myhub
'@'%' IDENTIFIED BY 'xyzzy#1';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT ALL PRIVILEGES ON `myhub_metrics`.* TO 'myhub
'@'%' IDENTIFIED BY 'xyzzy#1';
Query OK, 0 rows affected (0.00 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)

mysql> quit
```

Configure Apache for CMS

We are now going to tell Apache about the change of document root of the web server.

Modify **/etc/apache2/sites-available/hub *AND* /etc/apache2/sites-available/hub-ssl** replacing three instance of "/var/www" with "/www/myhub": follows

Change

```
DocumentRoot /var/www
...
<Directory /var/www>
...
</Directory>
...
<Directory /var/www/site/protected>
...
</Directory>
```

to

```
DocumentRoot /www/myhub
...
<Directory /www/myhub>
...
</Directory>
...
```

```
        <Directory /www/myhub/site/protected>
        ...
        </Directory>
```

then restart apache

```
# /etc/init.d/apache2 restart
```

Run CMS Installer

The Joomla/HUBzero installer will now appear on your website. Follow the instructions to get the basic site configured. Then disable the installation directory.

You can ignore the LDAP configuration screen as it is currently not used.

Press the "Install HUBzero Sample Data" button to have a basic site layout made [RECOMMENDED]

When you are done with the web installer you need to disable the web installation application:

```
# rm -fr /www/myhub/installation
```

Configure CMS

Some of the initial site configuration still needs be done manually.

```
# cp -p /www/myhub
/hubconfiguration.php-dist /www/myhub/hubconfiguration.php
```

Fill in the dozen or so parameters as needed and the core global site configuration is complete

```
<?php
class HubConfig {
    var $hubLDAPMasterHost = 'ldap://127.0.0.1';
    var $hubLDAPSlaveHosts = '';
    var $hubLDAPBaseDN = 'dc=myhub,dc=org';
    var $hubLDAPNegotiateTLS = '0';
```

```
    var $hubLDAPSearchUserDN = '';
    var $hubLDAPSearchUserPW = '';
    var $hubLDAPAcctMgrDN = 'cn=admin,dc=myhub,dc=org';
    var $hubLDAPAcctMgrPW = 'test';
    var $ipDBDriver = 'mysql';
    var $ipDBHost = 'db.nanohub.org';
    var $ipDBPort = '';
    var $ipDBUsername = 'geodb';
    var $ipDBPassword = 'ge0dbhub';
    var $ipDBDatabase = 'network';
    var $ipDBPrefix = '';
    var $hubShortName = 'myhub';
    var $hubShortURL = 'myhub.org';
    var $hubLongURL = 'http://myhub.org';
    var $hubSupportEmail = 'real@email.address';
    var $hubMonitorEmail = 'real@email.address';
    var $hubHomeDir = '/home/myhub';
    var $forgeName = 'myFORGE';
    var $forgeURL = 'https://myhub.org';
    var $forgeRepoURL = 'http://myhub.org';
    var $svn_user = 'hubrepo';
    var $svn_password = 'test';
}
?>
```

Be sure to use a real email address in hubSupportEmail and hubMonitorEmail, failure to do so will result in a failure of the hub to send out email properly.

Subversion

Install Subversion

```
# install --owner www-data --group www-
data --mode 0770 -d /opt/svn/tools
# touch /etc/apache2/svn.conf /etc/apache2/svn.bak
# chown www-data /etc/apache2/svn.conf /etc/apache2/svn.bak
```

The apache hub site configuration files are preconfigured to support this. Subversion repositories are generated dynamically by the addrepo script (installed later) and are included through the /etc/apache2/svn.conf file. URLs matching !^/tools/[^/]+/svn($|/) excluded from being processed by the CMS and are instead directly handled by the Subversion Apache module.

Test

```
# svnadmin create /opt/svn/tools/test --fs-type fsfs
# chown -R www-data.www-data /opt/svn/tools/test
# echo "<Location /tools/test/svn>
        DAV svn
        SVNPath /opt/svn/tools/test
        AuthType Basic
        AuthBasicProvider ldap
        AuthName "Test"
        AuthzLDAPAuthoritative on
        AuthLDAPGroupAttributeIsDN on
        AuthLDAPGroupAttribute owner
        AuthLDAPGroupAttribute member
        AuthLDAPURL ldap://localhost/ou=users,dc=myhub,dc=org
        Require ldap-group gid=apps,ou=groups,dc=myhub,dc=org
</Location&gt" > /etc/apache2/svn.conf
# /etc/init.d/apache2 restart
```

Be sure to the BASEDN in the above to match that used by your configuration.

Now browse to "/tools/test/svn" using an https connection and you should get prompted for a username and password, use the apps account you created earlier when you installed LDAP. You should see "svn - Revision 0: /".

Delete test file.

```
# echo "" > /etc/apache2/svn.conf
# rm -fr /opt/svn/tools/test
# /etc/init.d/apache2 restart
```

WebDAV

Configure WebDAV

The apache hub site configuration files are preconfigured to support this. Update the LDAP configuration to match the BASEDN of your site:

Edit **/etc/apache2/sites-available/hub-ssl**

```
        <Directory /webdav>
        ...

AuthLDAPURL ldap://localhost/ou=users,dc=myhub,dc=org?uid
        ...
        </Directory>
```

This enables webDAV for the /webdav directory space, rewriting the url to always be under the user's directory. The 'usermap' tool (see next section) is used to map files from the user's home directory into the /webdav space (and mapping ownership to www-data).

Then restart the apache webserver to enable your changes.

```
# /etc/init.d/apache2 restart
```

Test

```
# install --owner www-data --group www-
data --mode 0770 -d /webdav/home/apps
# touch /webdav/home/apps/test
```

Browse to your site's https /webdav address (e.g. https://myhub/webdav). You should get prompted for a username and password. Use the apps account. You should see a directory listing including the file "test".

Now test using a WebDAV client.

```
# apt-get install cadaver
```

```
# cadaver https://myhub.org/webdav
```

You will be prompted to accept self signed certificate (if it is still installed) and then to enter your username and password. Use the 'apps' account again to test. When you get the "dav:/webdav/>" prompt just enter "ls" and it should show the test file.

Finally clean up test case

```
# apt-get purge cadaver
# rm /webdav/home/apps/test
# rmdir /webdav/home/apps /webdav/home /webdav
```

Usermap

Install usermap

Install the WebDAV Usermap Filesystem

```
# apt-get install hubzero-usermap
```

Configure Usermap

```
# apt-get install autofs
```

Edit **/etc/auto.master** by adding the following line

```
/webdav/home /etc/auto.webdav --timeout=60
```

Edit/create **/etc/auto.webdav** so that it has the following content

```
* -fstype=usermap,user=www-data,allow_other :&
```

```
# /etc/init.d/autofs restart
```

Add **fuse** to the **/etc/modules** file so that it is loaded on startup.

This automounts a usermap filesystem of a users home directory inside of /webdav/home on demand. This version of the users home directory is owned and accessible to the user www-data which allows WebDAV to serve its contents.

Test

```
# touch /home/myhub/apps/test
# ls -l /webdav/home/apps
```

You should see a list of files in apps's home directory ("test") which will appear to be owned by www-data.www-data

```
# mount -l
```

You should see something like:
mount.usermap on /webdav/home/apps type fuse.mount.usermap (rw,nosuid,nodev,allow_other)

Finally clean up.

```
# umount -f /webdav/home/apps
# rm /webdav/home/apps/test
```

Troubleshooting

If the test doesn't work, check if the fuse kernel module is loaded

```
#  lsmod | grep fuse
fuse                   54176  0
```

If there is no output then try starting the kernel module manually

```
# modprobe fuse
```

Then try the test again

Trac

Configure Apache for Trac

```
# install --owner www-data --group www-
data --mode 0770 -d /opt/trac/tools
```

The apache hub site configuration files are preconfigured to support this. Update the LDAP configuration to match the BASEDN of your site:

Edit **/etc/apache2/sites-available/hub-ssl**

```
        <LocationMatch /tools/[^/]+/login>
                ...

AuthLDAPURL ldap://localhost/ou=users,dc=myhub,dc=org?uid?sub?(gid=*)
                ...
        </LocationMatch>
```

Then restart apache

```
# /etc/init.d/apache2 restart
```

Install Authentication Plugin

```
# apt-get install hubzero-trac-mysqlauthz
# /etc/init.d/apache2 restart
```

Test

```
# svnadmin create /opt/svn/tools/test --fs-type fsfs
# chown -R www-data.www-data /opt/svn/tools/test
# trac-admin /opt/trac/tools/test initenv "Test" "sqlite:db/trac.db" "
svn" "/opt/svn/tools/test"
# chown -R www-data.www-data /opt/trac/tools/test
```

Now browse to "/tools/test/wiki" using an https connection; you should see a default Trac project page.

Delete test data.

```
# rm -fr /opt/svn/tools/test
# rm -fr /opt/trac/tools/test
# /etc/init.d/apache2 restart
```

addrepo

Install

Install addrepo

```
# apt-get install hubzero-addrepo
```

Configure

The web process needs to be able to run a number of scripts as the "apps" user. To do this it is necessary to configure sudo to allow this:

Edit **/etc/sudoers** and add the following lines

```
www-
data        ALL=(apps)NOPASSW
D:/bin/bash /www/myhub/components/com_contribtool/scripts/*tool.php *
www-
data        ALL=(apps)N
OPASSWD:/usr/bin/expect /www/myhub
/components/com_contribtool/scripts/*tool.php *
www-data        ALL=NOPASSWD:/etc/init.d/apache2
```

Be sure to replace "/www/myhub" with the document root you are using for your hub.

iptables

iptables

```
# apt-get install hubzero-mw-firewall
```

HUBzero requires the use of iptables to route network connections between application sessions and the external network. The scripts controlling this can also be used to manage basic firewall operations for the site. If you use manage iptables with other tools you will have to make sure the rules in these scripts are maintained. /etc/mw/firewall_on and /etc/mw/firewall_off turn the HUBzero firewall on and off respectively. A link from /etc/rc.boot/firewall_on to /etc/mw/firewall_on causes the script to run at startup (this link was created for you). The basic scripts installed here block all access to the host except for those ports required by HUBzero (http,https,http-alt,ldap,ssh.smtp,mysql,submit,etc).

Maxwell Service

Install

```
# apt-get install hubzero-mw-service
```

Configure

```
# /usr/lib/mw/bin/mkvztemplate amd64 lenny
```

Test

```
# /usr/lib/mw/bin/maxwell_service startvnc 1 800x600 24
```

Enter an 8 character password when prompted (e.g., "testtest")

This should result in a newly create OpenVZ session with an instance of a VNC server running inside of it. The output of the above command should look something like:

```
Reading passphrase:
testtest
==================== begin /etc/vz/conf/hub-
session-5.0-amd64.umount ========================

Removing /var/lib/vz/root/1 :root etc var tmp dev/shm dev
==================== end /etc/vz/conf/hub-
session-5.0-amd64.umount ========================
stunnel already running
Starting VE ...
==================== begin /etc/vz/conf/1.mount ====================
=====
Removing and repopulating: root etc var tmp dev
Mounting: /var/lib/vz/template/debian-5.0-amd64-maxwell home apps
==================== end /etc/vz/conf/1.mount ======================
=====
VE is mounted
Setting CPU units: 1000
Configure meminfo: 2000000
```

```
VE start in progress...
TIME: 0 seconds.
Waiting for container to finish booting.
/usr/lib/mw/startxvnc: Becoming nobody.
/usr/lib/mw/startxvnc: Waiting for 8-byte vncpasswd and EOF.
1+0 records in
1+0 records out
8 bytes (8 B) copied, 3.5333e-05 s, 226 kB/s
Got the vncpasswd
Adding auth for 10.51.0.1:0 and 10.51.0.1/unix:0
xauth:  creating new authority file Xauthority-10.51.0.1:0
Adding IP address(es): 10.51.0.1
if-up.d/mountnfs[venet0]: waiting for interface venet0:0 before doing
NFS mounts (warning).
WARNING: Settings were not saved and will be resetted to original valu
es on next start (use --save flag)
```

```
# vzlist
      VEID      NPROC STATUS  IP_ADDR         HOSTNAME

         1          6 running 10.51.0.1       -
```

```
# openssl s_client -connect localhost:4001
```

This should report an SSL connection with a self signed certificate and output text should end with:

```
---
RFB 003.008
```

If you see this then you successfully connected to the VNC server running inside the newly created OpenVZ session.

Clean up

```
# /usr/lib/mw/bin/maxwell_service stopvnc 1
```

Which should give output similar to:

```
Killing 6 processes in veid 1 with signal 1
Killing 7 processes in veid 1 with signal 2
Killing 5 processes in veid 1 with signal 15
Got signal 9
Stopping VE ...
VE was stopped
==================== begin /etc/vz/conf/1.umount ====================
=====
Unmounting /var/lib/vz/root/1/usr
Unmounting /var/lib/vz/root/1/home
Unmounting /var/lib/vz/root/1/apps
Unmounting /var/lib/vz/root/1/.root

Removing /var/lib/vz/root/1 :root etc var tmp dev/shm dev
Removing /var/lib/vz/private/1: apps bin emul home lib lib32 lib64 mnt
 opt proc sbin sys usr .root
==================== end /etc/vz/conf/1.umount ======================
====
VE is unmounted
```

Maxwell Client

Install

```
# apt-get install hubzero-mw-client
```

Configure

Configure SSH to allow maxwell keyed clients to connect as root by copying the contents of /etc/mw/maxwell.key.pub into /root/.ssh/authorized_keys. This allows the maxwell client to run the maxwell service as root.

```
# mkdir -p /root/.ssh
# cat /etc/mw/maxwell.key.pub >>  /root/.ssh/authorized_keys
```

Copy the sample maxwell.conf file

```
#  cp /usr/lib/mw/maxwell.conf-dist  /etc/mw/maxwell.conf
```

Edit /etc/mw/maxwell.conf

```
mysql_host = "localhost"
mysql_user="myhub"
mysql_password="xyzzy#1"
mysql_db="myhub"

default_vnc_timeout=86400
session_suffix="L"

filexfe
r_decoration="
""filexfer_sitelogo { <h1><a
 href="http://myhub.org/" title="myHUB
 home page"><span>myhub.org
: online simulations and more</span></a></h1> }
filexfer_stylesheet http://myhub.org/templates/filexfer/upload.css"
"""
```

```
hub_name="myhub"
hub_url="http://myhub.org/"
hub_homedir="/home/myhub"
hub_template="hubbasic"
```

Test

```
# su www-data
$ ssh -i /etc/mw/maxwell.key root@localhost ls
The authenticity of host 'localhost (127.0.0.1)' can't be established.
RSA key fingerprint is e5:3c:7d:41:71:0b:0f:2a:0c:0e:bb:15:4d:e7:2f:08
.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (RSA) to the list of known host
s.
list of files
$ exit
#
```

VNC Client

Install

```
# apt-get install tightvnc-java
```

To confirm you have the correct version installed:

```
# dpkg --list | grep tightvnc-java
ii  tightvnc-java                            1.3.8-0+hubzero3
    TightVNC java applet and command line progra
```

You should see a "+hubzeroXX" suffix on the version, where XX is some number. This version includes SSL connection support and extra support for resizing the VNC session and a simple client action protocol used by the HUBzero VNC server.

If the wrong version is installed be sure you have added the hubzero package repository to you APT configuration. You may need to run "apt-get update" to get an up to date index of hubzero packages. After updating try installing tightvnc-java again.

vncproxy

Install

```
# apt-get install hubzero-mw-vncproxy
```

expire

Install

```
# apt-get install hubzero-mw-expire
```

telequotad

install

```
# apt-get install hubzero-mw-telequotad
```

App Container

Install

Apps (tools) on HUBzero run inside a Session Container. This is container's filesystem is rooted at /var/lib/vz/template/debian-5.0-amd64-maxwell. To install debian packages into the session container you must use chroot to root your filesystem on the Session Containers. Here we do this and install a number of Session Container related packages.

```
# chroot /var/lib/vz/template/debian-5.0-amd64-maxwell
# apt-get update
# apt-get upgrade
# apt-get install icewm
# apt-get install hubzero-icewm-config
# apt-get install hubzero-icewm-themes
# apt-get install hubzero-use
```

We use a modified version of icewm, to confirm you have the correct version got installed:

```
# dpkg --list | grep icewm
ii  icewm                                    1.2.35-1+hubzero1
    wonderful Win95-OS/2-Motif-like window manag
ii  icewm-common                             1.2.35-1+hubzero1
    wonderful Win95-OS/2-Motif-like window manag
```

You should see a "+hubzeroXX" suffix on the version, where XX is some number. This version fixes a bug when resizing the display when using vnc4server.

This fix should get applied to the upstream package someday, in which case we will be able to remove this version of the package and use version from the Debian Linux distribution.

Be sure to exit the chroot environment when you are done:

```
# exit
```

Configure

Users in the apps group can be granted permission to manage HUBzero Apps by placing them in the 'apps' group. This permission is granted through the sudo, and is configured by adding the following lines to /etc/sudoers:

```
%apps  ALL=NOPASSWD:/bin/su - apps
```

Workspace

Install

```
# apt-get install hubzero-app
# apt-get install hubzero-app-workspace
# hubzero-app setup
# hubzero-
app install --publish /usr/lib/hubzero/apps/workspace-1.0.hza
```

Configure

Go to the administrative web interface and select the Tools component

Click the on the parameters button and fill in values for host,username,password and database for the apps database. In the hub-in-a-box configuration these are the same values as the main HUBzero database.

Test

You should then be able to log in to the site and see the "Workspace" tool in the tool list and launch it in your browser.

Initialization

Setting Component Parameters

Rappture

Install

The Rappture application is install in the apps directory along with proper links.

```
# apt-get install hubzero-rappture
```

Session Installation

Rappture is used from inside a container and needs several other packages installed to allow use of all its features. This process has been simplified by using the hubzero-rappture-session with only contains the dependencies needed to pull in these other packages.

```
# chroot /var/lib/vz/template/debian-5.0-amd64-maxwell
# apt-get update
# apt-get upgrade
# apt-get install hubzero-rappture-session
```

This is also a good time to add some default paths to your session environment so that it doesn't need to be whenever someone logs in. Modify the /etc/profile file as follows:

Change

```
if [ "`id -u`" -eq 0 ]; then
  PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
else
  PATH="/usr/local/bin:/usr/bin:/bin:/usr/games"
fi
```

To

```
if [ "`id -u`" -eq 0 ]; then
  PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
else
  PATH="/apps/rappture/bin:/apps/bin:
/usr/local/bin:/usr/bin:/bin:/usr/games"
```

```
fi
```

Be sure to exit the chroot environment when you are done:

```
# exit
```

A workspace may need to be opened and closed a few times before the changes to the session template appear in a workspace.

Test

Rappture comes with several demostration scripts that can effectively test many parts of the package. These demonstrations must be copied to a user's home directory within a workspace before running.

```
$ mkdir examples
$ cp -r /apps/rappture/examples/* examples/.
$ cd examples
$ ./demo.bash
```

A window should open on the workspace showing that part of the demonstration. Close that window to see the next demonstration. Some demonstrations may need something inputted to work properly (such as the graphing calculator).

filexfer

Install filexfer

Install the filexfer package

```
# apt-get install hubzero-filexfer
```

Configure Apache for filexfer

Modify the hub site file at /etc/apache2/sites-available/hub-ssl

The apache hub site configuration files are not preconfigured to support this. Add the two highlighted lines as shown below. Note that the relative placement of these lines is important.

Edit **/etc/apache2/sites-available/hub-ssl**

```
<VirtualHost *:443>
        RewriteEngine   on
        RewriteMap      xlate           prg:/usr/lib/mw/bin/filexfer-
xlate
        ...
        ...
        ...
        <Directory /www/myhub>
                Options FollowSymLinks
                AllowOverride None
                Order allow,deny
                Allow from all

                RewriteEngine   On

RewriteRule     ^filexfer/(.*) ${xlate:$1|nothing} [P,QSA,L]
```

Then restart apache

```
# /etc/init.d/apache2 restart
```

Submit

Introduction

The submit command provides a means for HUB end users to execute applications on remote resources. The end user is not required to have knowledge of remote job submission mechanics. Jobs can be submitted to traditional queued batch systems including PBS and Condor.

Installation

```
# apt-get install hubzero-app-submit
# apt-get install hubzero-submit-server
# apt-get install hubzero-submit-distributor
```

Local Configuration

The behavior of submit is controlled through a set of configuration files. There are separate files for defining remote sites, staged tools, multiprocessor managers, legal environment variables, remote job monitors, and ssh tunneling.

Sites

Remote sites are defined in the file sites.dat. Each remote site is defined by a stanza indicating an access mechanism and other account and venue specific information. Defined keywords are

- [name] - site name. Used as command line argument (-v/--venue) and in tool.dat (destinations)
- venues - comma separated list of hostnames. If multiple hostnames are listed one site will chosen at random.
- tunnelDesignator - name of tunnel defined in tunnels.dat.
- siteMonitorDesignator - name of site monitor defined in monitors.dat.
- venueMechanism - possible mechanisms are ssh and local.
- remoteUser - login user at remote site.
- remoteBatchSystem - the possible batch submission systems include CONDOR, PBS, and LSF. SCRIPT may also be specified to specify that a script will be executed directly on the remote host.
- remoteBatchQueue - when remoteBatchSystem is PBS the queue name may be specified.
- remoteBatchPartition - slurm parameter to define partition for remote job
- remoteBatchPartitionSize - slurm parameter to define partition size, currently for BG machines.

- remoteBatchConstraints - slurm parameter to define constraints for remote job
- remoteBinDirectory - define directory where shell scripts related to the site should be kept.
- remoteScratchDirectory - define the top level directory where jobs should be executed. Each job will create a subdirectory under remoteScratchDirectory to isolated jobs from each other.
- remotePpn - set the number of processors (cores) per node. The PPN is applied to PBS and LSF job description files. The user may override the value defined here from the command line.
- remoteManager - site specific multi-processor manager. Refers to definition in managers.dat.
- remoteHostAttribute - define host attributes. Attributes are applied to PBS description files.
- stageFiles - A True/False value indicating whether or not files should be staged to remote site. If the the job submission host and remote host share a file system file staging may not be necessary. Default is True.
- passUseEnvironment - A True/False value indicating whether or not the HUB 'use' environment should passed to the remote site. Default is False. True only makes sense if the remote site is within the HUB domain.
- arbitraryExecutableAllowed - A True/False value indicating whether or not execution of arbitrary scripts or binaries are allowed on the remote site. Default is True. If set to False the executable must be staged or emanate from /apps.
- members - a list of site names. Providing a member list gives a layer of abstraction between the user facing name and a remote destination. If multiple members are listed one will be randomly selected for each job.
- state - possible values are enabled or disabled. If not explicitly set the default value is enabled.
- failoverSite - specify a backup site if site is not available. Site availability is determined by site probes.
- checkProbeResult - A True/False value indicating whether or not probe results should determine site availability. Default is True.
- restrictedToUsers - comma separated list of user names. If the list is empty all users may garner site access. User restrictions are applied before group restrictions.
- restrictedToGroups - comma separated list of group names. If the list is empty all groups may garner site access.
- logUserRemotely - maintain log on remote site mapping HUB id, user to remote batch job id. If not explicitly set the default value is False.

An example stanza is presented for a site that is accessed through ssh.

```
[cluster]
venues = cluster.university.edu
remotePpn = 8
remoteBatchSystem = PBS
remoteBatchQueue = standby
remoteUser = HUBuser
```

```
remoteManager = mpich-intel64
venueMechanism = ssh
remoteScratchDirectory = /scratch/HUBuser
siteMonitorDesignator = cluster
```

Tools

Staged tools are defined in the file tools.dat. Each staged tool is defined by a stanza indicating an where a tool is staged and any access restrictions. The existence of a staged tool at multiple sites can be expressed with multiple stanzas or multiple destinations within a single stanza. If the tool requires multiprocessors a manager can also be indicated. Defined keywords are

- [name] - tool name. Used as command line argument to execute staged tools. Repeats are permitted to indicate staging at multiple sites.
- destinations - comma separated list of destinations.
- executablePath - path to executable at remote site.
- restrictedToUsers - comma separated list of user names. If the list is empty all users may garner tool access. User restrictions are applied before group restrictions.
- restrictedToGroups - comma separated list of group names. If the list is empty all groups may garner tool access.
- remoteManager - tool specific multi-processor manager. Refers to definition in managers.dat. Overrides value set by site definition.
- state - possible values are enabled or disabled. If not explicitly set the default value is enabled.

An example stanza is presented for a staged tool maintained in the HUBuser account on a remote site.

```
[simulator]
destinations = cluster
executablePath = ${HOME}/apps/simulator/bin/simulator.ex
remoteManager = mpi
```

Multi-processor managers

Multiprocessor managers are defined in the file managers.dat. Each manager is defined by a stanza indicating the set of commands used to execute a multiprocessor simulation run. Defined keywords are

- [name] - manager name. Used in sites.dat and tools.dat.
- computationMode - indicate how to use multiple processors for a single job. Recognized

values are mpi, parallel, and matlabmpi. Parallel application request multiprocess have there own mechanism for inter process communication. Matlabmpi is used to enable the an Matlab implementation of MPI.

- preManagerCommands - comma separated list of commands to be executed before the manager command. Typical use of pre manager commands would be to define the environment to include a particular version of MPI amd/or compiler, or setup MPD.
- managerCommand - manager command commonly mpirun. It is possible to include strings that will be sustituted with values defined from the command line.
- postManagerCommands - comma separated list of commands to be executed when the manager command completes. A typical use would be to terminate an MPD setup.
- mpiRankVariable - define environment variable set by manager command to define process rank. Recognized values are: MPIRUN_RANK, GMPI_ID, RMS_RANK, MXMPI_ID, MSTI_RANK, PMI_RANK, and OMPI_MCA_ns_nds_vpid. If no variable is given an attempt is made to determine process rank from command line arguments.
- environment - comma separated list of environment variables in the form e=v.
- moduleInitialize - initialize module script for sh
- modulesUnload - modules to be unloaded clearing way for replacement modules
- modulesLoad - modules to load to define mpi and other libraries

An example stanza is presented for a typical MPI instance.

```
[mpich-intel32]
preManagerCommands = . ${MODULESHOME}/init/sh, module load mpich-
intel32
managerCommand = mpirun -machinefile ${PBS_NODEFILE} -np NPROCESSORS
```

The token NPROCESSORS is replaced by an actual value at runtime.

Environment variables

Legal environment variables are listed in the file environmentwhitelist.dat. The objective is to prevent end users from setting security sensitive environment variables while allowing application specific variables to be passed to the remote site. Environment variables required to define multiprocessor execution should also be included. The permissible environment variables should be entered as a simple list - one entry per line. An example file allowing use of a variable used by openmp is

```
# environment variables listed here can be specified from the command
line with -e/--env option.

OMP_NUM_THREADS
```

Monitors

Remote job monitors are defined in the file monitors.dat. Each remote monitor is defined by a stanza indicating where the monitor is located and to be executed. Defined keywords are

- [name] - monitor name. Used in sites.dat (siteMonitorDesignator)
- venue - hostname upon which to launch monitor daemon. Typically this is a cluster headnode.
- tunnelDesignator - name of tunnel defined in tunnels.dat.
- remoteUser - login user at remote site.
- remoteMonitorCommand - command to launch monitor daemon process.

An example stanza is presented for a remote monitor tool used to report status of PBS jobs.

```
[cluster]
venue = cluster.university.edu
remoteUser = HUBuser
remoteMonitorCommand = ${HOME}/SubmitMonitor/monitorPBS.py
```

Tunnels

In some circumstances access to clusters is restricted such that only a select list of machines is allowed to communicate with the cluster job submission node. The machines that are granted such access are sometimes referred to as gateways. In such circumstances ssh tunneling or port forwarding can be used to submit HUB jobs through the gateway machine. Tunnel definition is specified in the file tunnels.dat. Each tunnel is defined by a stanza indicating gateway host and port information. Defined keywords are

- [name] - tunnel name.
- venue - tunnel target host.
- venuePort - tunnel target port.
- gatewayHost - name of the intermediate host.
- gatewayUser - login user on gatewayHost.
- localPortOffset - local port offset used for forwarding. Actual port is localPortMinimum + localPortOffset

An example stanza is presented for a tunnel between the HUB and a remote venue by way of an accepted gateway host.

```
[cluster]
venue = cluster.university.edu
venuePort = 22
gatewayHost = gateway.university.edu
```

```
gatewayUser = HUBuser
localPortOffset = 1
```

Remote Configuration

For job submission to remote sites via ssh it is necessary to configure a remote job monitor and a set of scripts to perform file transfer and batch job related functions. A set of scripts can be used for each different batch submission system or in some cases they may be combined with appropriate switching based on command line arguments. A separate job monitor is need for each batch submission system. Communication between the HUB and remote resource via ssh requires inclusion of a public key in the authorized_keys file.

Job monitor daemon

A remote job monitor runs a daemon process and reports batch job status to a central job monitor located on the HUB. The daemon process is started by the central job monitor on demand. The daemon terminates after a configurable amount of inactivity time. The daemon code needs to be installed in the location declared in the monitors.dat file. The daemon requires some initial configuration to declare where it will store log and history files. The daemon does not require any special privileges any runs as a standard user. Typical configuration for the daemon looks like this:

```
siteDesignator     = "cluster"
monitorRoot        = "/home/HUBuser/SubmitMonitor"
qstatCommand       = "/usr/pbs/bin/qstat -u HUBuser"
monitorLogLocation = "logs"
```

The directory defined by the combination of monitorRoot and monitorLogLocation needs to be created before the daemon is started. A sample daemon used for PBS batch systems is listed below.

```
#!/usr/bin/env python
#
# Copyright (c) 2004-2010 Purdue University All rights reserved.
#
# Developed by: HUBzero Technology Group, Purdue University
#               http://hubzero.org
#
# HUBzero is free software: you can redistribute it and/or modify it u
nder the terms of the
# GNU Lesser General Public License as published by the Free Software
```

```
Foundation, either
# version 3 of the License, or (at your option) any later version.
#
# HUBzero is distributed in the hope that it will be useful, but WITHO
UT ANY WARRANTY;
# without even the implied warranty of MERCHANTABILITY or FITNESS FOR
A PARTICULAR PURPOSE.
# See the GNU Lesser General Public License for more details.  You sho
uld have received a
# copy of the GNU Lesser General Public License along with HUBzero.
# If not, see .
#
# GNU LESSER GENERAL PUBLIC LICENSE
# Version 3, 29 June 2007
# Copyright (C) 2007 Free Software Foundation, Inc.
#
# ----------------------------------------------------------------------
--
#  monitorPBS.py
#
#  script which monitors the PBS queue and reports changes in job stat
us
#
import sys
import os
import os.path
import select
import time
import popen2
import re
import signal

siteDesignator     = "pbsHost"
monitorRoot        = os.path.join(os.sep,'home','pbsUser','Submit','pb
sHost')
qstatCommand       = "/usr/pbs/bin/qstat -u pbsUser"
monitorLogLocation = "logs"
monitorLogFileName = "monitorPBS.log"
historyFileName    = "monitorPBS.history"

logFile            = sys.stdout
historyFile        = None
activeJobs         = {}
updates            = []
```

```
def cleanup():
   global historyFile

   if historyFile:
      historyFile.close()


def sigGEN_handler(signal, frame):
   global siteDesignator

   cleanup()
   log("%s monitor stopped" % (siteDesignator))
   sys.exit(1)

def sigINT_handler(signal, frame):
   log("Received SIGINT!")
   sigGEN_handler(signal, frame)

def sigHUP_handler(signal, frame):
   log("Received SIGHUP!")
   sigGEN_handler(signal, frame)

def sigQUIT_handler(signal, frame):
   log("Received SIGQUIT!")
   sigGEN_handler(signal, frame)

def sigABRT_handler(signal, frame):
   log("Received SIGABRT!")
   sigGEN_handler(signal, frame)

def sigTERM_handler(signal, frame):
   log("Received SIGTERM!")
   sigGEN_handler(signal, frame)

signal.signal(signal.SIGINT, sigINT_handler)
signal.signal(signal.SIGHUP, sigHUP_handler)
signal.signal(signal.SIGQUIT, sigQUIT_handler)
signal.signal(signal.SIGABRT, sigABRT_handler)
signal.signal(signal.SIGTERM, sigTERM_handler)


def openLog(logName):
   global logFile

   try:
      logFile = open(logName,"a")
```

```
   except:
      logFile = sys.stdout


def log(message):
   global logFile

   if message != "":
      logFile.write("[%s] %s\n" % (time.ctime(),message))
      logFile.flush()


def openHistory(historyName,
                accessMode):
   global historyFile

   if accessMode == "r":
      if os.path.isfile(historyName):
         historyFile = open(historyName,accessMode)
      else:
         historyFile = None
   else:
      historyFile = open(historyName,accessMode)


def recordHistory(id):
   global updates
   global activeJobs

   historyFile.write("%s:%s %s %s\n" % (siteDesignator,str(id),activeJ
obs[id][0],activeJobs[id][1]))
   historyFile.flush()
   updates.append(str(id) + " " + activeJobs[id][0] + " " + activeJobs
[id][1])


def getCommandOutput(command,
                     streamOutput=False):
   child = popen2.Popen3(command,1)
   child.tochild.close() # don't need to talk to child
   childout = child.fromchild
   childoutFd = childout.fileno()
   childerr = child.childerr
   childerrFd = childerr.fileno()

   outEOF = errEOF = 0
```

```
   BUFSIZ = 4096

   outData = []
   errData = []

   while 1:
      toCheck = []
      if not outEOF:
         toCheck.append(childoutFd)
      if not errEOF:
         toCheck.append(childerrFd)
      ready = select.select(toCheck,[],[]) # wait for input
      if childoutFd in ready[0]:
         outChunk = os.read(childoutFd,BUFSIZ)
         if outChunk == '':
            outEOF = 1
         outData.append(outChunk)
         if streamOutput:
            sys.stdout.write(outChunk)
            sys.stdout.flush()

      if childerrFd in ready[0]:
         errChunk = os.read(childerrFd,BUFSIZ)
         if errChunk == '':
            errEOF = 1
         errData.append(errChunk)
         if streamOutput:
            sys.stderr.write(errChunk)
            sys.stderr.flush()

      if outEOF and errEOF:
         break

   err = child.wait()
   if err != 0:
      log("%s failed w/ exit code %d" % (command,err))
      if not streamOutput:
         log("%s" % ("".join(errData)))

   return  err,"".join(outData),"".join(errData)


if __name__ == '__main__':

   if monitorLogFileName != "stdout":
      openLog(os.path.join(monitorRoot,monitorLogLocation,monitorLogFi
```

```
leName))

   log("%s monitor started" % (siteDesignator))

   sleepTime = 10
   pauseTime = 5.
   maximumConsectutiveEmptyQueues = 30*60/sleepTime

   openHistory(os.path.join(monitorRoot,historyFileName),"r")
   if historyFile:
#  lPBS:6760 R
#  --------------------
      records = historyFile.readlines()
      for record in records:
         colon = record.find(":")
         if colon > 0:
            jobState = record[colon+1:].split()
            id = jobState[0]
            status = jobState[1]
            stage  = "Simulation"
            activeJobs[id] = (status,stage)
      historyFile.close()

      completedJobs = []
      for activeJob in activeJobs:
         if activeJobs[activeJob][0] == "D":
            completedJobs.append(activeJob)

      for completedJob in completedJobs:
         del activeJobs[completedJob]

   openHistory(os.path.join(monitorRoot,historyFileName),"a")
   consectutiveEmptyQueues = 0

   toCheck = []
   toCheck.append(sys.stdin.fileno())
   while 1:
      updates       = []
      currentJobs   = {}
      completedJobs = []

      delayTime = 0
      while delayTime  0:
         updateMessage = str(len(updates)) + " " + siteDesignator + ":
" + ":".join(updates)
         sys.stdout.write("%s\n" % (updateMessage))
```

```
         sys.stdout.flush()

      del updates

      if consectutiveEmptyQueues == maximumConsectutiveEmptyQueues:
         cleanup()
         log("%s monitor stopped" % (siteDesignator))
         sys.exit(0)
```

File transfer and batch job scripts

The simple scripts are used to manage file transfer and batch job launching and termination. Examples of scripts suitable for use with PBS are listed here.

File transfer - input files

receiveinput.sh - receive compressed tar file containing input files required for the job. The file .__fileTimeMarker is used to determine what files should be returned to the HUB.

```
#!/bin/sh
#
rm -rf $1
mkdir $1
exitStatus=$?

if [ $exitStatus -eq 0 ] ; then
   cd $1
   exitStatus=$?

   if [ $exitStatus -eq 0 ] ; then
      tar xvzf -
      exitStatus=$?

      touch .__fileTimeMarker
      sleep 1

      date +"%s" > $2
   fi
fi

exit $exitStatus
```

Batch job script - submission

submitbatchjob.sh - submit batch job using supplied description file. If arguments beyond job working directory and batch description file are supplied an entry is added to the remote site log file. The log file provides a record relating the HUB end user to the remote batch job. The log file should be placed at a location agreed upon by the remote site and HUB.

```
#!/bin/sh
#
cd $1
exitStatus=$?

if [ $exitStatus -eq 0 ] ; then
   case $2 in
      *.pbs)
         JOBID=`qsub $2`
         exitStatus=$?
         if [ $# -gt 2 ] ; then
            logRecord=`date`
            shift 2
            while [ $# -gt 0 ] ; do
               logRecord=${logRecord}'\t'$1
               shift 1
            done
            logRecord=${logRecord}'\t'${JOBID}
            echo -e ${logRecord} >> pbslog
         fi
         ;;
      *)
         echo "Invalid job class $2"
         exitStatus=23
         ;;
   esac
fi

if [ $exitStatus -eq 0 ] ; then
   echo ${JOBID}
else
   echo "-1"
fi
exit $exitStatus
```

File transfer - output files

transmitresults.sh - return compressed tar file containing job output files.

```
#!/bin/sh
#
cd $1
exitStatus=$?

if [ $exitStatus -eq 0 ] ; then
   tar czf - `find . -newer .__fileTimeMarker -not -name . -print`
   exitStatus=$?
fi

exit $exitStatus
```

Batch job script - termination

killbatchjob.sh - terminate the batch job

```
#!/bin/sh
#
case $2 in
   PBS)
      qdel $1
      exitStatus=$?
      ;;
   *)
      echo "Invalid job class $2"
      exitStatus=23
      ;;
esac

exit $exitStatus
```

File transfer - cleanup

cleanupjob.sh - remove job specific directory and any other dangling files

```
#!/bin/sh
#
rm -rf $1
exitStatus=$?

exit $exitStatus
```

Access Control Mechanisms

By default tools and sites are configured so that access is granted to all HUB members. In some cases it is desired to restrict access to either a tool or site to a subset of the HUB membership. The keywords restrictedToUsers and restrictedToGroups provide a mechanism to apply restrictions accordingly. Each keyword should be followed by a list of comma separated values of userids (logins) or groupids (as declared when creating a new HUB group). If user or group restrictions have been declared upon invocation of submit a comparison is made between the restrictions and userid and group memberships. If both user and group restrictions are declared the user restriction will be applied first, followed by the group restriction.

In addition to applying user and group restrictions another mechanism is provided by the boolean keyword arbitraryExecutableAllowed in the sites configuration file. In cases where the executable program is not pre-staged at the remote sites the executable needs to be transferred along with the user supplied inputs to the remote site. Published tools will have their executable program located in the /apps/tools/revision/bin directory. For this reason submitted programs that reside in /apps are assumed to be validated and approved for execution. The same cannot be said for programs in other directories. The common case where such a situation arises is when a tool developer is building and testing within the HUB workspace environment. To grant a tool developer the permission to submit such arbitrary applications the site configuration must allow arbitrary executables and the tool developer must belong the system group submit.