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.